Last Modified: 24-MAY-96

```
+----------------------------------------------------------------------mh<DoH!>-+
| M______.  A______.  C______.    H____.   A______.  C______.  K____.     |
| )    :\ )    :\ )    :\    )  / /\ )    :\ )    :\ )  / /\/\ |
| /      :\/  /\  :\/  /\__/_  /  /__\:\/  /\  :\/  /\__/_/  /_/  / |
|/  / \  :\  / \  :\  \_\  :\ /  _____  :\  / \  :\  \_\  :\  .__.  :\ |
|\___\__/______\  \_____________/ \___\  /______\  \________________\  \___\|
+-----------------------------------------S.ssSS88$$s----S-S----,ssSS88$$8s-----+
 Contributors: AX1P, Filbert,         S8   '"~$$~    S8 8S   8$$8S"~~~$8S
 Observer, Maddog Hoek, oleBuzzard,   S8            S8   8S  S8$$      ~8S
 Armchair Hacker, ArcAngel, Nganon   S8$.ssSS88$$s  S8$   $8S  S8$$      $8S
 DATE -=> 04-MARCH-1996            S8$    '"~$$~  S8$.sss.$8S  S8$      $8S
 ______kn0wledge phreak BBS______  S8$$          S8$$'"~'"$$8S  S8$     $$8S
  New home of MHFAQ 719.578.8288  S8$$          S8$$      $$8S  S8s  $   $$8S
   WebSite: http://iti2.net/k0p  S8$$$         S8$$$  ver  $$$8S  S8s $$ s$$$8
      e-mail: k0p@iti2.net      ~$~           ~$~   2.0   ~$~   ~S8$$$Ss~$~
===========================================================================$$$
$====
```

SECTION I: SOFTWARE DEPROTECTION/'CRACKING'

SECTION II: SYSTEMS HACKING

SECTION III: SYSTEMS HACKING

SECTION IV: PHREAKING

SECTION V: MAC UNDERGROUND RESOURCES

SECTION VI: MAC HACK TIDBITS

==
===

00. Introduction to the MacHackFAQ v2.0

Welcome to the MacHackFAQ v2.0! This thing has been awhile in the making, but
I think I've revamped it to a level that I can work with. I'd like to be able
to put out new FAQs at least every three months, the greatest determinor of
that will be the volume of article submissions. To start this FAQ off, heres
some House Cleaning issues:

Contributors--Contributors this month are: Observer, Maddog Hoek, Voyager,
ArcAngel, AX1P, Spooty, Filbert, The Jackal, Mark O'Connel, Nganon, me
(oleBuzzard). Thanx to everyone who contributed. My apologies to anyone who
contributed that I failed to acknowledge.

MacHack FAQ Header--This additions Header was created by Maddog Hoek. If you
are an ASCII artist, and would like to submit a Header for upcoming FAQs
please contact me.

Home of the FAQ--An html versions of the FAQ can be found at kn0wledge phreak
WWW page. Text versions of MacHack FAQ can be found at kn0wledge phreak WWW
page or kn0wledge phreak BBS

Submissions, Corrections, Praises, Complaints, Suggestions--If you want to contact me regarding any of the following, please feel free to e-mail me. Please label your subject as one of the five subjects.

Addresses--I kept saying you could contact me, I supposed you'd like to know where.

oleBuzzard's E-mail Address: k0p@iti2.net
kn0wledge Phreak WWW Page: http://iti2.net/k0p
kn0wledge phreak BBS: 719-578-8288

SECTION I: SOFTWARE DEPROTECTION/'CRACKING'
----

01. What is MACSBUG?

MacsBug is an acronym for Motorola advanced computer systems deBugger. It is an assembly-language-level debugging tool for the Macintosh and Power Macintosh computers. MacsBug was written by Motorola (creator of the 68000 series chip) to aid programmer's in development of Macintosh software. The versatility of MacsBug also makes it a very useful tool for software deprotection.

02. Where can I find MacsBug?

MacsBug can be found at the Apple Corporation FTP Support Site:

http://www.support.apple.com/pub/Apple%20SW%20Updates/US/Macintosh/Utilities

03. How do you use MacsBug?

The answer comes from Observer in an Original piece written for the FAQ:

Macsbug for Fun and Profit
Macsbug is an awesome program published by Apple and available for free. It's used by programmers to debug their programs, and crackers to help them in their work. Macsbug (MB) is what's called a "low-level debugger." This is because it works at a very low level--in other words, looking at the actual instructions being executed by the computer. Currently, the latest version of MB is 6.5.2.

Installing Macsbug is easy. Drop it in your System Folder and restart. Don't double click on it, don't put it in the Extensions folder, don't try to give it more memory--just put it in the System Folder and let it be. The next time you restart, the message "Debugger installed" will accompany your normal Welcome to Macintosh message. This confirms that Macsbug is loaded.

To stop processing and enter Macsbug (called breaking into Macsbug), press the interrupt button on your Mac. This is a small button with a circle on it. Inside the circle is a little squiggly line that looks sort of like an EKG (sometimes it's just a circle, though). It will often be accompanied by an adjacent small button with a triangle in it. This is the reset button.

Anyway, press the interrupt button, and Macsbug will appear. If your computer is one of those without hardware reset/interrupt buttons, press cmd-power. (cmd-ctrl-power is the equivalent of the reset button.)

Macsbug makes you look very cool when you use it. This is because it looks like sheer hell to anyone who doesn't know how to interpret what it gives you. What does it give you? Here's an ASCII picture of a MB screen: (view in Monaco)

```
________________________________________________________________________
|    SP      |                                        |
|  nnnnnn    |                                        |
|            |                                        |
| CurApName  |                                        |
| SimpleText |                                        |
|            |                                        |
| 32-bit RM  |            [previously executed        |
| SR SmxNZvc |             instructions, plus         |
|            |           output generated by          |
| D0 nnnnnn  |             your commands, show up     |
| [...]      |           here]                        |
| D6 nnnnnn  |                                        |
| D7 nnnnnn  |                                        |
|            |________________________________________|
|            | [proc name]                 ; will branch        |
| A0 nnnnnn  |   +nnnn     nnnnnn   BCC.S         | 641A        |
| [...]      |   +nnnn     nnnnnn * MOVE.L        | 2008        |
| A7 nnnnnn  |   +nnnn     nnnnnn   CLR.W         | 4267        |
|____________|__________________________________|_____________|
```

Whoa! What the HELL is all this stuff? (And who in the world uses it?) Basically, unless you're using assembly language on the Mac (as a programmer or cracker, for example), you don't need to know what all this stuff means. For the benefit of those who care, however, here you go. (Other people, skip down to the next section.)

SP
Stack Pointer. Not too important except for programmers/crackers.

CurApName
The name of the currently running application. This is NOT (NOT NOT NOT)not the frontmost application! Many times it will not be. To ensure that an application will be running when you break into macsbug, hold down

one of its menus.

32-bit RM
Indicates whether you are in 32 or 24 bit memory mode (on any modern Mac will always be 32)fairly and whether you're using Real Memory or Virtual Memory.

D0-D7, A0-A7: Data and address registers on the 680x0 chip, where data is sometimes stored.

[proc name]
The name of the subprogram which is being executed, or "no procedure name" if none is available. If ResEdit/Resorcerer tell you the name of a subprogram is something line "<Anon_17>," MB just says "no procedure name."

; will branch
If the next instruction to be executed (the instruction directly below the procedure name) is a branch, this will pop up and say whether or not the branch will occur.

+nnnn
The offset within the current procedure of the instruction on that line.

nnnnnn
The absolute address in memory of the instruction on that line.

*
Shows up if there's a breakpoint set on an instruction. Unless you're setting breakpoints, you won't get any of these.

BCC.S, MOVE.L, etc.
The next assembly instructions which will be executed.

641A, 2008, etc.
The hex equivalent of these instructions.

And that's about it. There are lots of worthwhile things you can do in Macsbug without understanding all this stuff, though.

es
Exit to Shell. Attempts to quit the current program and go back to the finder. If you crash and use this, it's best to restart the computer ASAP.

rs
ReStart. Useful if you crash and can't use es, but don't want to do a hardware restart. Better than turning the computer off, because it unmounts mounted volumes.

rb
ReBoot. Same as rs, but doesn't unmount mounted volumes. This makes it more or less the same as turning the computer off and then back on, or hitting a hardware reset button.

help <topic | command>
Displays help for the specified topic or command. To see a list of topics, just type "help".

Base 10 <-> Base 16 (hex) <-> ASCII conversion
Enter a number preceded by # for decimal, $ for hex, or in single quotes (i.e. 'q') for ASCII. Hit return. What pops up is the hex, decimal and ASCII equivalent! Nifty, eh?

Error ID lookup
Crashed and want to know just what an error -43 is? Break into Macsbug and type:

```
error #(error ID in base 10)
```

and Macsbug will tell you what the error means.

A calculator!
Macsbug can perform mathematical operations, such as *, +, -, /, even between number systems!

You can also do some fun stuff with Macsbug:

sw menuflash [hexadecimal number 1-FFFF]
Sets the number of times a menu item flashes when selected. If you set this over 50 or so, be prepared to be very patient!

Strobe light
Type "swap". Macsbug will say "Display will be swapped after each trace or step." Now type "s 20" and hit return. Ooooh!! Aaaah!! Make the number bigger if you like, but be patient... Type swap again to end the process.

And in case it ever comes up in Trivial Pursuit:
The name Macsbug has nothing to do with Macs. It is an acronym for Motorola Advanced Computing Systems deBUGger. If Apple had called their computers Donuts, Macsbug would still be called Macsbug. (Motorola comes in, for those who don't know, because Motorola makes the 680x0 chips which were the heart of every Mac until the PowerPC, which is still made by Motorola.)

For Andy Ihnatko's typically unique spin (I mean that kindly, Andy) on Macsbug, check out the last page of the Feb 96 MacUser. If you're a Mac programmer and want to know how to use Macsbug to examine your programs, check out _Debugging Macintosh Software with Macsbug_, by Othmer and Straus.

For information on how to use Macsbug itself, Apple publishes a manual which costs about $30.

04. How can I use MacsBug to crack software?

"How do I get blahblahware to stop asking me to register?"
(Also known as, "Will someone give me a crack to blahblahware?")

Intro...
Cracking software is a huge topic--not always difficult, but one with many different aspects, all of which can be important. This is just the first step down a long road, and I urge anyone interested in truly learning about cracking to check out the "Further Reading" section at the bottom. Also, the first two appendixes (glossary and assembly reference) aren't meant as afterthoughts but as important parts of the text. Use them. Appendix 3 is useful if you want Resorcerer (which you do).

Background...
Anyone who's written a few real Mac applications (or one big one) in Pascal, C, or any similar language is a good candidate to become a Mac cracker. However far down from there you rank yourself, is how much harder it's going to be for you to crack software. Try if you like, but knowing how to program is useful if you want to modify programs.

If you're freaked out about assembly language, don't be; a decent programmer in Pascal or C can acquire a fluency in assembly fairly easily. All your friends from the Toolbox exist in assembly, just with an underscore ("_") before their names. And we call them traps, rather than calls. But other than that they're pretty much the same. And lots of cracking is just changing branches, like changing conditions in an "if" statement. Nothing too hairy, right?

People generally write programs in what's called a high-level language, a language that's far from what the computer actually does but is easy for a human to remember and work with. HyperTalk is a very high-level language. Pascal and C are another notch or two down the line. In order for the computer to run programs written in these high-level languages, you need a compiler. This is a program which translates what you've written in Pascal (gibberish to the computer), into assembly language, the specific instructions which the CPU will execute to run your program. So when you open a program and look at its CODE resources, you're looking at some representation of the actual instructions the computer follows to run that program.

The Hunt...
Note I said some representation. If you're using ResEdit, all you'll see is the code in hexadecimal. This doesn't do you much good. To view it as its assembly code equivalent, either spring for Resorcerer (a $256 ResEdit done

right), or get the ResEdit CODE Editor, which is free and publicly available. Once you install the resources in the CODE Editor into your ResEdit application, when you open a CODE resource, you'll see something like this (and also get some new menus):

```
Offset   Addr    Opcode     Operand      Comment
=========================================================
```

Here's what this all means:

Offset
The line number in bytes, counting from the beginning of the CODE resource segment

Addr
The line number, counting from the beginning of the current procedure/subprogram

Opcode
The assembly instruction to execute

Operand
Data which accompanies the instruction (parameters)

Comment
Misc. info on a line of code, plus hex representation of the line

All this exists in Resorcerer as well, just with slightly different names. To toggle between viewing absolute and relative offsets in Resorcerer, press cmd-2 while viewing a CODE resource.

Go to the "Modules" (Routines in Resorcerer) menu. There you'll find a list, in the order they exist in the code, of all of the procedures in that code segment. (Happy Resorcerer users will have this menu alphabetized.) Find a program which has more than anon1, anon2, etc. Procedure names are a huge help to a cracker, because let's say you want to remove a registration dialog box--which catches your eye more, "DoRegDialog," or "anon36?"

So you have your program. Let's say what's annoying you is that it always shows a dialog which you can't dismiss for a few seconds, until it enables the OK button.

Go look at the program's DLOG resources and find the dialog you want to avoid. If it isn't there, check out the WIND resources as well. Convert the dialog/window's ID number into hex. If you can't do this manually, Resorcerer can do it for you, or else find one of the many shareware calculators that has the capability. Also, TI-85 owners can just punch go into the mode settings and set it to use hex. Never thought that thing would come in handy, did you?

Anyway. Search for this value in the code, just a few lines before a call to the _GetNewDialog trap. (Cmd-G in Resorcerer, or hold down option when opening the CODE resource in ResEdit and use ResEdit's search tools.) Here's a sample from an actual application, whose nag dialog is DLOG ID #9990=$2706:

```
move.w    #$2706,-(sp)
clr.l     -(sp)
pea       -$0001
_GetNewDialog
```

What's this doing? It's MOVEing the hex number $2706 to "sp." This is the Stack Pointer, a place where things are stored temporarily--typically parameters passed to a procedure or function, and afterwards what it returns. Sure enough, the next line is:

```
movea.l   (sp)+,a4
```

This is where we move the DialogPtr given to us by _GetNewDialog, off of the stack pointer and put its address in register A4. (We know GetNewDialog returns a DialogPtr because we bought the Inside Mac CD while we were doing Mac programming in a high-level language. I wasn't kidding when I said Mac programming experience would help.)

The Kill...
OK, so now we know where the dialog is loaded. And, because we've used dialogs in a higher-level language before, we know that other toolbox calls--ModalDialog and CloseDialog for example--tend to accompany a GetNewDialog call. Further, the problem we wish to overcome is that it stops for a few seconds before enabling the OK button. This implicates another likely accomplice, HiliteControl, which is used to enable and disable dialog items.

Let's say the programmer was a jerk and left the subprogram names in the code. Maybe the subprogram you found the dialog in is called "DoNagBox." If it's this obvious, you could try NOP'ing the entire DoNagBox subprogram. Note that while this is easy in Resorcerer, it is very difficult in ResEdit.

Maybe that doesn't work. Maybe that makes the program crash. OK, time to try something else. While the nag box is open, break into Macsbug (read about that in another section of the FAQ) and type "atb closedialog". This will cause Macsbug to interrupt processing when a call to the _CloseDialog trap is made. Dismiss the nag dialog, and poof, you're in Macsbug. Use the "t" command to step through the code, through the subprogram which holds the _GetNewDialog for the nag box. When you hit an "rts," keep going--the next line will be the line after the line which calls the nag subprogram. Here's a little diagram:

```
                /-> doNagBox
               /      [other assembly]
[assembly]    /        move.w $2706, -(sp)
             /         _GetNewDialog
jsr doNagBox /         [more assembly]

[more assembly]<---\   _CloseDialog
                    \  [still more assembly]
                     \-rts
```

We reach "jsr doNagBox," which sends us off to the doNagBox subprogram. This puts up a dialog and then closes it when we hit a button. When all this has been done, we're returned to the line of code immediately following the "jsr doNagBox" line. Just like any other language.

We could NOP the "jsr doNagBox," but that tends to be asking for trouble; any parameters passed to or received from the subprogram are left wandering around, which will probably cause a crash. What we should look for are branches, probably beq or bne. Is there one of these above the jsr which skips down just a few lines past the jsr? If so, try changing the condition of this branch (such as beq->bra).

Other Techniques

The idea of looking for a dialog's ID is one which frequently works. However, there are other limitations you might want to overcome. Here are some ideas for other program limitations:

Only works for x minutes, then quits

Look for the _TickCount trap (hex A975) in the code--this is the most common method of doing this. Something else to watch for is _ExitToShell, (hex A9F4), which MAY be the way the program quits itself. If the subprogram names are in the code, look especially hard at anything resembling "eventloop," "mainloop," etc.

Only works for a week

Look for the _SecondsToDate (hex A9C6) trap, and a branch a while after it. Also, if a dialog pops up to tell you to register, look for the ID of this dialog.

Only lets you play the first x levels

Several possibilities here. If a dialog appears when you reach a higher level, the easiest is to search for the dialog ID in the code. If it quits, look for _ExitToShell. If you absolutely can't find what you're looking for, search for the highest possible level number in the code. (If you can only play levels 1-4, search for $0004.) If this shows up in or near some form of cmp, you may have struck paydirt.

Practice, Practice, Practice

With just a few months of practice, you'll be surprised at how many things

you can crack in less than an hour. Here are some things you can try looking at, in order of difficulty: (easiest->hardest)
Relax 1.0 (any shareware site)
GraphicConverter 1.7.7 /1 (ditto)
Warcraft 1.0
Net Watchman demo (ftp.aggroup.com) (don't worry about printing)
GopherGolf 2.0.7 (shareware again)
DragStrip 1.2.4

(Note: Some of these are commercial software. These cracks should only be attempted on software you own, and for your own convenience.)

Appendix 1: Glossary

Branch:
Each command in assembly has an offset, essentially a line number. Branching to an offset sets the PC to the specified offset and then continues execution normally.

Byte, word, long word:
The most common data sizes. Use monaco for the table below:

	Bits	Hex digits	Range (decimal)
Byte	8	2	0-255
Word	16	4	0-65535
LWord	32	8	0-4294967295

These can be halved to alter the range to include negative values. So a byte can also be used to go from #-127 to #127, a word from #-32767 to #32767, and so on. In a long word (for example) this is accomplished by going from $0 to $7FFF (#0-#32767) normally. $8000 is then equal to #-32767, up to $FFFF=#-1. The same system is used for the other data sizes as well.

Flags:
There are five status flags: Z, C, N, V, X. These keep track of the results of operations. Conditional branches such as bne and beq check the flags to decide whether or not to branch.
Z: Zero flag. Set if the result of an operation is zero, or if two compared values are equal. Cleared otherwise.
C: Carry flag. Set if the a math operation produced a digit carry (i.e. $FF+$1)
N: Negative flag. Set if the result of a math operation is negative, or the most significant (rightmost) bit in a number is true.
V: Overflow flag. Set if an operation's result can't be held in the data provided (such as $FF+$1 in a byte). Not too common.
X: Extended flag. Used for precision in math operations. Also not too common.

Hexadecimal:
Usually referred to as hex. This is base 16. Our number system is base 10 (aka decimal), which means each column is ten times the previous one. In hex,

you start with the ones column, then you have a sixteens column, then a 256's column, and so on. Hex is just like our normal system, except you count to 15 before going to the next place. The extra 6 numbers you need for this are provided by the letters A-F. So counting in hex goes like this:
1,2,3,4,5,6,7,8,9,A,B,C,D,E,F,10,11,12,13,14,15,16,17,18,19,1A,1B,1C,1D,1E,1F,20,21...
The signs # and $ are used to indicate decimal (base 10) and hexadecimal, respectively. So #10=$A; (#15+#1)=($F+$1)=$10; #255=$FF; and so on.
Two hexadecimal digits are equivalent to eight bits, or one byte.

Registers:
680x0 chips have 16 registers, which are places to hold data (essentially a variable in higher-level languages). These are divided into 8 data registers, labeled D0-D7, and 8 address registers, labeled A0-A7. Each register can hold a long word. The address and data registers are themselves identical, but there are commands which can be used on address registers which cannot be used on data registers.

Subprogram/Subroutine/Procedure/Function:
Used more or less interchangeably. If used specifically, they mean the same thing they would in a high-level language.

Appendix 2: Quick Assembly Instruction Reference
This is a brief description of the most common commands in assembly language. There are many others however, and anyone seriously wanting to learn how to crack will soon need more than this. See the "further reading" section for suggestions.
Suffixes: .b, .w, .l
Indicates that the suffixed instruction will apply to a Byte, Word, or Long word, respectively. So cmp.b will compare two bytes.

add
ADDs two values, and stores the result in the second operand. The Z flag is set if the result was zero, cleared otherwise.

beq
Branch if EQual. Branches if Z flag is set. 67 hex.

bne
Branch if Not Equal. Branch if the Z flag is clear. 66 hex.

bra
BRanch Always. Move PC to the indicated offset and continue. 60 hex.

clr
CLeaR. Sets its operand to zero.

cmp

CoMPares two values. If the values are equal then the Z flag is set. Otherwise it is cleared.

jsr/rts
Jump SubRoutine. Exactly like calling a procedure or function in a high-level language: sets PC to the subprogram's address, but first puts the PC's current value on the stack. When the specified subprogram is completed, the rts ("ReTurn from Subroutine") command will be used to return to where the subprogram was called.

link/unlk
LINK/UNLinK. Generally used to create local variables for subprograms. (Link creates, unlink disposes at end of subprogram.)

move
MOVEs the first operand into the second. When you see something like (A2), it means that the data stored in the address held in A2 is being used. A2 without the parentheses means the actual data held in A2.

nop
No OPeration. Useful for simply deleting code without changing the location in memory of other code. 4E71 hex.

sub
SUBtract. Same as add, but subtracts the first operand from the second.

Appendix 3: Ordering Resorcerer, a cracker's best friend
The single-copy price of Resorcerer is US $256 (decimal!). We also offer quantity, reseller, and educational discounts at anywhere between 20% and 50% off of the above price. Please call us for more information and a quote.

Our mailing address is:
Mathemaesthetics, Inc.
P.O. Box 298
Boulder, CO, 80306-0298
Phone: (303) 440-0707
Fax: (303) 440-0504

Internet: resorcerer@aol.com

Appendix 4: FURTHER READING
Surprise surprise, a few pages aren't enough to teach you assembly language. For more information, check out these sources...
Files by The Shepherd and Vassal
Each of these guys has written a much bigger file on Mac cracking. The Shepherd's is the larger one and better for the beginner (and a great file in general), Vassal's offers more specific technique tips. I used the Shepherd's file as a reference for the assembly reference section here.

Basic MacCracking files
I've written a few files which describe how to crack specific programs. Of course I'm biased, but I think these are all very helpful to beginners, especially since they were written as I learned things myself.

Fantasm's help files
Fantasm is an assembly language development program, for the sickos who actually create whole programs in assembly language. While using the program itself has been shown to cause severe social problems, it comes with six large files written to teach someone how to write assembly language. These aren't something anyone serious about this stuff should pass up.

Debugging Macintosh Software with Macsbug
Macsbug in invaluable to a cracker. I would be shot if I took the space to describe how to use it here, but it's not that hard to figure out. What is hard is discovering how to use it in the context of a Macintosh (i.e. where is the event record that _waitnextevent just got?), and this book tells you all of that.

Macsbug Reference and Debugging Guide
Apple's Macsbug documentation, plus EXCELLENT assembly tutorial. Another one serious folks shouldn't miss out on.

05. What are some other useful MacsBug related resources?

DBugr 1.2.1...........Puts a floating bomb on your desktop that you can click

on at any time to enter macs bug. Widely available.
http://vsl.cnet.com Search: 'macsbug'

Break Before..........Break into MacsBug on the very first instruction of the

INIT code of ANY extension you choose. Widely
available.
http://vsl.cnet.com Search 'macsbug'

Debugger F-Key........Drop into the debugger. Recognizes MacsBug, TMON, The
Debugger, and ABZmon. Will also recognize any new
debuggers that follow Apple's debugger protocol as
documented in the "MacsBug Reference and Debugging
Guide." http://vsl.cnet.com Search 'macsbug'

Cool MacsBug Tricks...Cool things you can do with MacsBug.
http://www.biddeford.com/~benyc/Macsbug.html

Tips for MacsBug......Place to obtain and submit MacsBug programming tips.
http://www.scruznet.com/~crawford/Computers/macsbug.html

SECTION II: SYSTEMS HACKING
----

06. What are some general techniques for defeating Macintosh Security?

Here are a few:

* Restart a system with the Shift-key down to disable extensions.
* Restart with the built-in ROM Disk available on some Macs. Hold: Command-Option-x-o during boot-up.
* Boot from a floppy. Even if floppy startup has been disabled, you should be able to force it by holding down the command-option-shift-delete key combo to boot the floppy. This key combo won't let the internal hard drive mount.

07. What are some general tools for defeating Macintosh Security?

MUST HAVES for defeating Secured Macs are Keystroke Recorders, file wipers and the System 7.5 Disk Tools.

Keystroke Recorders--Keystroke Recorders are normally Control Panels, which when activated, will record every keystroke made on a system. In many cases the log containing all of the Keystroks is stored in a covert place for later retreival. A few keystroke recorders are:

Invisible Oasis.......http://wheel.dcn.davis.ca.us/~sean/hack/hack.html
MacLife Insurance.....http://vsl.cnet.com Search: 'maclifeinsurance'
SuperSave 1.x.........http://vsl.cnet.com Search: 'super save'

File Wipers--File wipers are utilities that can remove a file from a Hard Disks by physically writing over it. Many files are protected against deletion by the prevention of routines which allow their altering. File wipers can circumvent this protection because they don't perform the routines involved in altering a file, instead they just write over the file with null data. As a result the file is eliminated and thereby rendered NON-FUNCTIONAL. This makes them a very valuable in defeating Macintosh security. File wipers have the ability to wipe: locked file, protected files, running programs, the system folder, themselves, anything. A few file wipers are:

Burn 2.2.............http://vsl.cnet.com Search: 'Burn'
Flame File v1.5.8....http://vsl.cnet.com Search: 'flamefile'
Obliterate v1.1......http://vsl.cnet.com Search: 'Obliterate'
The Eraser 2.0.0.....ftp://ftp.euro.net/Mac/info-mac/disk/eraser-20.hqx

System 7.5 Disk Tools--System 7.5 Disk Tools contains a Finder and Mini-System on a single 1.44 HD Floppy thereby alllowing you to boot from the

Floppy Drive. The 7.5 Disk Tools are a part of the System 7.5.

08. How can I Hack At Ease?

There are numerous ways to Hack At Ease. Here are a Few:

Programmer's Switch--Hit the programmer's switch (see section on MacsBug) and type: G FINDER. This should break you out of At Ease and leave you in the Finder. Once you're in in the the Finder you've pretty much hacked the protection. If you want to polish the hack (cover your tracks, find passwords, etc.) here's some advanced steps you can take:

* If you know that your sysadmin keeps logs Copy the system folder to the hard drive. Rename the original system folder. Reboot without At Ease. When you are done, put the real system folder back and delete the second one.

* If you aren't concerned about logs, just move the At Ease Preferences out of the System Folder: Extensions folder and reboot. Remember to put them back when you are done.

* Install one of the aforementioned Keystroke recorders. Wait a few days and check the logs from the recorder. You should have the administrator password!

Crashing the System--Another Hack for At Ease lies in Crashing the system it's running on. Just keep opening applications until all the RAM is consumed. On older versions of At Ease, a dialogue box will appear that asking you if you would like to quit At Ease to free up RAM. Click yes!

Null Password--Open the file System Folder:At Ease:At Ease Preferences with MSWord or any ther text editor. Look for the string "MFDR\   ]". Delete everything between "\" and "]". Save the changes and you have a null password. Now you can go to At Ease Setup and change the password to whatever you want!

OEM Hack---The following directions are excerpted from the At Ease Administration Manual from the Section: 'What do I do if I forget my Administrator Password?'

> If you forget the At Ease administrator's password, follow the directions below instead of those in the manual. If your startup disk is locked, you'll first need to run the Unlock application on the AT Ease 2.0 Utilities disk to unlock the start-up disk. Consult the manual for information about the Unlock application.
>
> 1. Start up your computer from another startup disk.
> 2. Open the System Folder of your usual startup disk.

3. Open the At Ease Items folder inside your System Folder.
4. Drag the At Ease Preferences file into the trash.
5. Hold down the Option key while you choose Empty Trash from the Special menu.
6. Restart from your usual startup disk.
7. Open the At Ease Setup for Workgroups application.

Note: If you are using an AppleShare server volume as the At Ease disk, your setups may not appear until you reset the At Ease disk to this server volume.

8. Reconnect to the server volume and use the At Ease Disk command to reselect the volume.

Note: Make sure you use the information on the server instead of replacing it with the information on the startup disk.

9. Add a new password and clue.
10. Make sure the following options set correctly:

* Allow Remote Administration checkbox
* Lock Startup Volume checkbox

11. Turn At Ease back on.
12. Quit At Ease Setup for Workgroups."

09. How can I use DisEase to Hack At Ease?

DisEase is a fairly powerful utility for Hacking At Ease. It allows you to manipulate At Ease, break out of At Ease, decode passwords, any number of things that would render At Ease useless. The only problem is on most At Ease protected system you are prevented from finder or floppy disk access, thereby preventing you the ability to run DisEase in the first place. In these situations, follow the above steps for breaking into the Finder, you can then access DisEase and use it to decode the Administrator password.

10. Where can I find DisEase?

DisEase 1.0.......ftp://ftp.eskimo.com/u/a/adrenal/mac/DisEase.hqx
DisEase 3.0.......http://www.tyrell.net/~ibs/Hackr/Cracking/DisEase3.0.sit.hqx

Also you can contact the Author. macpants@aol.com

11. How can I Hack FoolProof?

FoolProof is Macintosh security scheme that uses driver level and Systems Folder protection to prevent against bypassing. Driver Level protection is protection written to the Driver Level of the Hard Disk. At this level, the drive can not be mounted without envoking the protection. This condition will continue to exist as long as the Driver remains intact. Here are some methods of defeating it.:

IMPORTANT NOTE--The FoolProof extension, among other things, intercepts the Restart & Shutdown calls from the System and makes sure to disable any external boot device whenever a Restart or Shutdown is called. To defeat this, when you Restart YOU HAVE TO DO HARD RESTARTS (ctrl-opt-del). When you do hard Restarts no calls are made to the System, and the System is restarted without locking or protecting anything. So be sure to do HARD RESTARTS when hacking FoolProof.

Floppy Boot--As detailed in the beginning of this section, Boot from a floppy with command-option-shift-delete held down. This will prevent the Protected Driver from loading. Once the System is loaded you may need to use a Disk mounting utility to Mount the Hard Drive. Once the drive is mounted, Move the FoolProof Extensions and Prefs out of the System folder and Restart. FoolProof should be disabled.

<>

```
--------------------------------------------------------------------------
Transfer interrupted!
```