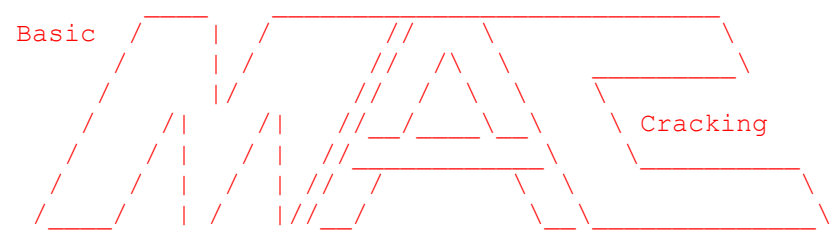

Part IV: Net Watchman revisited
by The Observer, released 12/19/95

Hello, Net Watchman.
My name is The Observer.
You killed my father.
Prepare to DIE.

OK, maybe this quote from The Princess Bride is a bit dramatic. But that's the attitude I came back to Net Watchman with. It had stumped me before, but now that I could get around worth a damn in Macsbug, I was ready to try again.

If you're following this series, you may remember that Net Watchman's demo has three limitations:
1. It only runs for 10 minutes
2. It deletes the files containing all the setup you'd done
3. It won't print

Basic MacCracking II got past limitation 1. Number 2 was the most serious of the remaining two, so that's where I decided to start. One reason I couldn't get this before was that the only mention of "Delete" in the code was a call to the _Delete OS trap, which isn't in Inside Mac. I had tried NOP'ing this and other parts of its subprogram, but doing this always caused an error when making a new file.

The Hunt
There are, intelligently enough, two points at which the demo deletes its files: when you close a file, and when you quit. Given that CODE 2 contains a subprogram called Terminate (which we remember from part II is involved in quitting), this was an easy place to start.

We find Terminate's absolute offset in CODE 2, 4D52. Then we open the program and break into Macsbug. To break at Terminate, we need the address of CODE 2 in Net Watchman's application heap. To see this, enter "hd". (Make sure that CurApName, in the middle of the vertical bar on the left, is Net Watchman! (Or at least "Net Watchm...") It frequently does not correspond to the frontmost application. To get it to be the frontmost app, click in the menu bar a few times.)

A lot of junk will scroll down now. All you want is CODE 0002--look for it. Here are a few actual lines, including one with CODE 2: (your results will vary)
 Displaying the “Net Watchman 2.1 Demo” heap at 0131FE90

```
     Start    Length      Tag  Mstr Ptr Lock Prg  Type   ID   File      Name
  • 0131FED0 00000100+04   N
  • 0131FFE0 00000242+02   R   0131FFC4   L       CODE  0001  189C
  • 01320230 00000044+00   R   0131FFC0   L
  • 01320280 00000044+00   R   0131FFBC   L
[...]
  • 01320580 00000014+00   N
  • 013205A0 0000655A+0A   R   0131FF90   L   P   CODE  0002  189C
```

Once you find CODE 2's line, go to the leftmost hex number (in the sample above, 013205A0). In my experience, this number is constant until you quit the application, and I suspect this is actually the case.

Let's say you get 013205A0 as well. To break at offset 4D52 in CODE 2, tell Macsbug:
br 013205A0+4D52
And Macsbug will reply:
Break at 013252F2 (Terminate) every time

013252F2 is just the memory address resulting from the addition of 013205A0 and 4D52. You could have done this yourself, but Macsbug does a fine job of it. So now we're going to break into Macsbug when Terminate begins to execute. An alternate way to have done this would be to type "br Terminate". This method is easier, but doesn't work when the subprogram names aren't in the assembly.

Sure enough, we hit cmd-q, and poof, Macsbug appears. Enter "t" to start stepping through the code. In just a little bit we come to this line:
+00030 012B09B2   JSR   CloseAlertFile | 4EAD 092A

When it's the next line up for execution (marked with an asterisk), enter "s". This sends you into the CloseAlertFile subprogram. Once you're there, start using "t" again, to keep things moving along. Almost immediately, we come to:
+00004 012B3574   JSR   QBKillAll | 4EAD 03DA

Killing might well have some connection to deleting, so let's use s to step into QBKillAll. We go through it until we find another likely candidate to step into:
+0000C 012B96E6   JSR   QBKill   ; 012B9706 | 4EBA 001E

Ever progressing, QBKill contains this line:
+00046 012B974C   JSR   KillNBP | 4EAD 097A

And finally, we can step through KillNBP in its entirety:
     +00000 012B87A6   LINK   A6,#$FFFC | 4E56 FFFC
     +00004 012B87AA   MOVEM.L   D7/A4,-(A7) | 48E7 0108
     +00008 012B87AE   MOVEA.L $0008(A6),A4 | 286E 0008

```
     +0000C 012B87B2   MOVEQ      #$00,D7                              | 7E00
     +0000E 012B87B4   CMPI.W     #$0001,$0010(A4)                     | 0C6C
0001 0010
     +00014 012B87BA   BNE.S      KillNBP+00074          ; 012B881A    | 665E
     +00074 012B881A   TST.L      $0022(A4)                            | 4AAC
0022
     +00078 012B881E   BEQ.S      KillNBP+00084          ; 012B882A    | 670A
     +00084 012B882A   MOVEM.L    -$000C(A6),D7/A4                     | 4CEE
1080 FFF4
     +0008A 012B8830   UNLK       A6                                   | 4E5E
     +0008C 012B8832   RTS                                             | 4E75
```

Fun as going deeper and deeper here has been, there's unfortunately a problem; we're never calling _Delete. Perhaps there's another way of doing it that is here? We can't tell. So let's just keep going through the code. We pass the end of KillNBP, then QBKill. QBKillAll, however, now proceeds to call QBKill--and in it, KillNBP--a second time. Net Watchman only deletes one file when it quits. This makes it seem fairly unlikely that any of these is responsible for the deletions.

So now we're back in CloseAlertFile, and there's nothing to do but keep trudging on:

```
+00008 012B3578   MOVE.W     #$FFFF,-(A7)                           | 3F3C FFFF
+0000C 012B357C   CLR.L      -(A7)                                  | 42A7
+0000E 012B357E   JSR        WOAlertRsrc            ; 012B685E      | 4EBA 32DE
```

We could go through WOAlertRsrc and see if it calls _Delete eventually, but its name doesn't immediately sound like it does anything with files. We'll keep it in mind in case we get stuck later, though.

```
+0001C 012B358C   CLR.W      (A7)                                   | 4257
+0001E 012B358E   PEA        -$0100(A6)                             | 486E FF00
+00022 012B3592   MOVE.W     -$13B0(A5),-(A7)                       | 3F2D EC50
+00026 012B3596   JSR        'CODE 0002 183E'+02FA0                 | 4EAD 02BA
```

Oooh! There! The subprogram with _Delete is unnamed (those using Resorcerer will see it as <Anon_11>). And it occurs in CODE 2 at offset 2FA0. This looks like what we've been waiting for. To make sure, we'll step into it. As we expected, it contains the following line:

```
+02FB6 012AEBE6   _Delete                          ; 001A8366    | A009
```

The Kill

So now we know Terminate offset 26 is calling CloseAlertFile, which in turn calls an unnamed subprogram to delete our files. As playing the with the unnamed subprogram (CODE 2+2FA0) itself gives us errors we have to get rid of the code which calls the subprogram. We saw in the second code snippet above that it's at offset 26 in CloseAlertFile. Hopping into Resorcerer, we find what it looks like when it's not being executed:

```
+26
jsr       $02BA(a5)            ; 4EAD 02BA
```

The actual code contains no reference to what it calls, which is why I couldn't get it without the Macsbug knowledge I picked up in writing part III. In any case, let's try NOP'ing this line.

And it works! Not only when we quit, but whatever you're doing when you close a file apparently calls this procedure as well. Pretty slick. So the file deletions, as it turns out, wasn't such a tricky problem after all.

Printing Problems
Now we have to coax it to print. First, we have an observation from part II which turned out to be fairly useless. We find a subprogram in CODE 4 called PrintWindow. If we were to follow a cmd-p keystroke with Macsbug, we'd find that it does engage this subprogram to print. We find the following snippet in Resorcerer:

```
beq.s     PrintWindow+$4E
move.w    #$000C,-(sp)
jsr       $014A(a5)
addq.l    #$2,sp
move.l    #-$30000000,-(sp)
```

Conditional branches are of course a great way to do or not do something depending on whether or not the program is registered, so let's use Macsbug ("br PrintWindow") to find out where the jsr is going.

```
+00042 00A6085C   BEQ.S      PrintWindow+0004E     ; 00A60868   | 670A
+00044 00A6085E   MOVE.W     #$000C,-(A7)                       | 3F3C 000C
DEL+00048 00A60862   JSR        DoNoteAlert                       | 4EAD 014A
+0004C 00A60866   ADDQ.L     #$2,A7                             | 548F
+0004E 00A60868   MOVE.L     #$D0000000,-(A7)                   | 2F3C D000
0000
```

Bingo! It's the one calling DoNoteAlert. Let's swap the conditional branch-if-equal BEQ for a straight, always-branch BRA.

No alert box pops up, unsurprisingly. But nothing prints either! Apparently the call to the printing routine is missing from the code. So it looks like cracks for this file were just destined to come out bit by bit; I still can't overcome the printing limitation.

However, here are some fun things to look at in Net Watchman (aka things I tried that didn't work).
Follow PrintWindow all the way through, including GetPrintHandle and UnloadNonPrintSegs.
Check out the ValidatePeek and ValidatePeekWork procedures in CODE 13. See how VPW pops up the registration dialog box?

OK, so just two fun things. Assembly does get boring sometimes.

Another twofer:
Shareware program called Personal Log. Asks you to register it each time you open it. NOP'ing the procedure HandleRegisterCopy (where the registration box pops up--you can find this easily by now, right?) causes the program to crash. But is this procedure always called? Where is it called? Once you find this out with Macsbug, check to see if there's a branch near it that might be the registered/unregistered branch...

For an extra point, get the "about" box to display the registration info of

your choice (this one is kinda silly).

Conclusion junk...
So I still can't get this stupid program to print. It will happen. Some day, it will happen. The fact it doesn't delete its settings files and runs for 10 minutes makes it a functional program, though. Printing is more or less fluff.

I have a pretty good idea for Part V, but I don't know if it will work out or not. And I'd hate for this to grind to a crawl just because I have to spend all my time combing the net for demos. Hell, I'll even extend an offer of fame to those who submit/suggest things to try--want your name in a Basic MacCracking file? Get those demos in. So when you think crippled software, think Observer! And mail me at one of these addresses. :-)
Observer on Kn0wledge Phreak (719-578-8288), The Keep, or an407599@anon.penet.fi.

Yet another thing to read!
While I can't endorse it over any others, I found a copy of Tibet Mimar's "Programming and Designing with the 68000 Family." It's a pretty good reference for straight assembly, but can get wordy sometimes. Some things it talks about are neat to know, if not that useful (did you know there was a 68008, an 8-bit 68000?) It's from '91, so it only goes up to the 68030. But it was a kick to actually see all these things I've been working with these past few months actually printed in a book, and a dedicated assembly reference has really been a help.