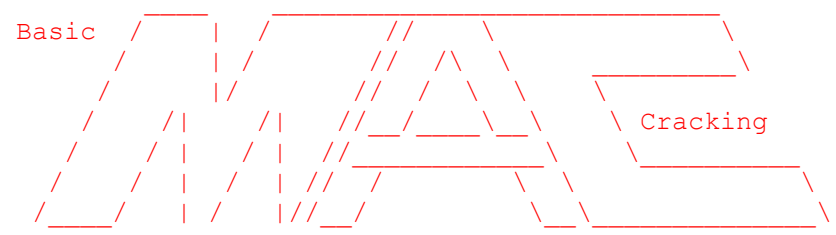


Part III: Whimsy 1.1.1, a Hermes External
by The Observer, released 12/7/95

Welcome back to Basic MacCracking! A bit after I finished part II, oleBuzzard mentioned that no one had been able to crack some Hermes externals, and that people would probably be interested if I did. The first one of these I've gotten through through was Whimsy 1.1.1.

What is it?
Whimsy pops up random messages at logon, logoff, and welcome times during a Hermes session. However, until you cough up the $10 registration fee, it will sometimes slip in a notice that your sysop hasn't registered this copy of Whimsy, and would you please email him and suggest he do so.

Stupid program.
Yeah, that's what I thought too. Let's take a peek at the code with Resorcerer.

Uh-oh. This is a Hermes external, and all it has is a big XHRM resource. If we can't get beyond a hex representation of this thing's code, we're in trouble. So let's copy the entire XHRM into a CODE resource, which will let us use Resorcerer's code editing tools on it.

The pasting works OK, except for one thing. Who knows whether this is *actually* working? You can take any random data, paste it into a CODE resource, and Resorcerer will show you its assembly equivalent. How do we know this is good CODE? We just don't. [OK, OK--in the case of Whimsy, we do. Whimsy's subprogram names are present in its assembly code. When I first started looking at Hermes externals though, the one I looked at didn't have the names, so it wasn't obvious.]

If Whimsy was not the first Hermes external we had looked at, we'd now need to see how a Hermes external is born. Is it with some Hermes interpreter and scripting language? Or a normal language with special compiler? What's the deal? A logical place to find such information was the Hermes tech support BBS, Olympus. Its number is 206-643-2874. A call here proved very helpful--I found several files that told me all I need to know, and would even let me write my own externals if I wanted to.

It turns out you can use Pascal, C, whatever the hell you want to make a Hermes external. Then, according to a project file which accompanied some source I'd downloaded, I saw it was compiled into a resource of type XHRM. I compiled the sample code and saved the result. Then I changed the destination resource type to CODE. The same data was produced. So it turns out all the scary XHRM is, is a renamed CODE resource. Now that that's settled, we can

actually get down to business.

Oooh, CODE!!
So now we have a happy CODE resource. We look at the nicely-preserved subprogram names. We notice one named RegRoutineAnnoying. The nag note is certainly annoying. What a nice place to start.

Sure enough, RegRoutineAnnoying has a call to _TickCount, and then a call to _Random:

```
movea.l   $0008(a6),a3
clr.l     -(sp)
_TickCount            ; TB trap
move.l    (sp)+,$38A8(a4)
clr.w     -(sp)
_Random               ; TB trap
move.w    (sp)+,d7
```

Random is a very good way to make things happen randomly, and TickCount provides a nice seed for generating pseudorandom numbers. Reasonable pair to be seeing together. So now we know where it's deciding to display or not display the nag message, right?

Not quite. Let's look at the full line with _Random in Resorcerer:

```
_Random               ; TB trap
 |   A861   |    ®a
```

What this is is a call to Random. The two vertical lines separate the sections of Resorcerer's screen into code, hex, and data. First is code--the decoded assembly or toolbox calls. Next comes the hex trap number, and finally its ASCII equivalent. We see A861 is _Random's trap number. We search the code for this, and fine one instance besides the one above, sitting in RegRoutineCool. Which one is it that is responsible for displaying the annoying message? (One could make a fair guess, but let's be scientific about this.) Macsbug offers an easy way to break at toolbox traps. We open Hermes, break into Macsbug manually, and enter:

```
atb a861
```

Macsbug says back:

```
A-Trap Break at A861 (_Random) every time
```

As you might have guessed, we will now break into Macsbug each time _Random (or, to be specific, the trap A861) is called. Now let's log in and see what happens.

Boom! Macsbug breaks with the message:

```
A-Trap break at 00D7069C RegRoutineAnnoying+00016: A861 (_Random)
```

So here we are sitting in the RegRoutineAnnoying subprogram, offset 16. To check if perhaps RegRoutineCool is called as well, we enter "g" so Macsbug will let the Mac go on with its business. No more breaks. So RegRoutineAnnoying is what we want to kill.

To find the place(s) RegRoutineAnnoying is called, is somewhat annoying. What I do is use the oddly placed "Print >  To Text File..." command to save the

code to a text file. Then I open this in Word, and look for the string "annoying." The expectation is to find a jsr pointing to it. As it happens, though, it's only referred to in one place, CheckCode offset 01F0:

```
lea       RegRoutineAnnoying,a1
```

## Who's Calling?

Resorcerer *says* that it's called there. It even has a branch to RegRoutineAnnoying. Looks pretty convincing. But LEA isn't a branch, or even a jump--it stands for Load Effective Address, and all it's doing here is placing the address of the RegRoutineAnnoying subprogram into register a1. Despite Resorcerer's say-so, CheckCode 01F0 is NOT calling RegRoutineAnnoying. (Take this as further evidence that computers are stupid, and a reminder not to trust them unquestioningly.) So what this line is doing is storing the routine's address, to call later. But where?

To find out, we go back into Macsbug, and break once more at the _Random trap. Once we break, we follow the procedure until rts, which sends us back from where we were called. (This is wisdom gained first-hand from part I.) We emerge from RegRoutineAnnoying into ShowScreen offset AE. This is the move.l in the code below:

```
movea.l   (a0),a0
jsr       (a0)
move.l    a3,(sp)
move.w    #$0002,-(sp)
```

indicating the jsr(a0) just before it is what called RegRoutineAnnoying. Therefore, we were correct that lea is just loading RegRoutineAnnoying's address, to be called later.

So this is at least one spot where RegRoutineAnnoying is called. Chances are it's the only one, but since it's using the LEA system there could be more. We could test for this by breaking at RegRoutineAnnoying (type "br ", hit cmd-D, choose RegRoutineAnnoying, hit return), but we still might not find all of them. On this assumption (that we can't get all the places it's called), the ideal solution will be to modify the RegRoutineAnnoying subprogram itself.

## Innie or Outie?

Notice I keep saying subprogram. This is a neutral term for both function (which returns a value) and procedure (which is supposed to not return anything). RegRoutineAnnoying has to be one of these, which brings up two possibilities--either it's self-contained, and so NOP'ing it will be an effective crack. OR, it returns its decision as to whether or not to display the nag note, to be acted on elsewhere. My guess, since nothing is moved off the stack pointer after it's called, is that it's a self-contained procedure. But I'm not confident enough in my assembly yet to be making a statement like that. [As it turns out later, I was part wrong on this anyway.] So an easy test to find out what it really is, I decided, would be to NOP pretty much the whole RegRoutineAnnoying subprogram:

```
link      a6,#$0000
nop
nop
...
nop
```

```
unlk      a6
rts
dc.b      $92,'RegRoutineAnnoying'
dc.w      #$0000
```

So now the entire subprogram is just link and unlink. We now go into Hermes, set a break on A891 just for good measure, and log in.

Whoa! It breaks in RegRoutineAnnoying, and all the code is still there. What the hell? This is something that took me a little bit to realize, and I still keep on forgetting to do it sometimes. Hermes doesn't give a shit what's in your CODE resource. What it looks at is the XHRM. So quit Hermes, take your modified CODE and paste it back in the XHRM. Now re-open Hermes and try again.

It looked to me like this worked. It looked so much like it worked, that I even sent it to oleBuzzard and asked him to try it out. The next day he mailed me back, though--something was wrong.

Experimental Error!

While testing the crack, I had been using only one test file. If the file came up, good; if Whimsy's nag came up, bad. Only two possibilities. oleBuzzard tried it with more than one file though, and quickly realized that it kept picking the same file. When I heard about this I was pretty annoyed with myself for not realizing that no call to _Random meant no randomness. I had been wrong that it was self-contained. No, RegRoutineAnnoying got data out of itself somehow, though without the stack pointer. Global variables are my best guess.

So it was back to work. What to do, what to do? My first idea was to check out all the branches in RegRoutineAnnoying and ShowScreen, to see if there was any consistent difference between when it showed a file message or the nag. Annoyingly, there wasn't. To examine this for yourself, break at _Random and step through, recording each branch through the end of ShowScreen. Deciding to play around nonetheless, I changed one and got the somewhat gratifying result of either seeing a file displayed, or nothing. No more nags, but still not perfect.

My next idea was to try and get Whimsy to let me register it with an otherwise illegal code. An little under an hour or so of dicking around got me the chance to enter a code, but things were so fucked up to get there that I couldn't make it work any further.

Well duh.

Finally, it hit me--RegRoutineAnnoying, and RegRoutineCool. Both have _Random's, but one is defined as annoying and one as cool. So we go back to CheckCode 01F0:

```
lea       RegRoutineAnnoying,a1     |     43FA  F156    |   C˙ÒV
```

This says RegRoutineAnnoying because Resorcerer is interpreting the actual instructions given the location of RegRoutineAnnoying in the code. What it *really* looks like is:

```
lea      *-$0EA8,a1
```

(To change between these, hit cmd-2, for "Routine-Relative Offsets" in the Code menu.)

So we're moving backwards from where we are (absolute offset 281E) to absolute offset 1976. Doing the math, we find that 281E-0EA8 does in fact equal 1976. Cool, nothing weird there. However, the hex code for the line is "43FA F156". -0EA8, as should be apparent, does not appear here. So I had to fiddle a bit to see what was going on.

If we set the second word of the hex (F156) to FFFF, it points backwards just one. Whoops. So apparently the amount it branches up is the difference between FFFF and what you give it. (Branching down, you just give the change in offset.) RegRoutineCool is at absolute offset 1A2E. We're at 281E and want to go to 1A2E. 281E-1A2E finds the difference between the two offsets: 0DF0. Subtract this from FFFF (65536), and get F20E. We put this in the second word of the line's hex code, save, and close the code window. Sure enough, when we open it again, the line reads:

```
lea       RegRoutineCool,a1
```

So now, instead of going to RegRoutineAnnoying, ShowScreen will call RegRoutineCool. Incidentally, in case you're not sick of hex by now, this is a system called two's complement, which is used to have a long word (4 bytes) go from -32767 to 32767, rather than zero to 65536. The system goes from 0000 to 7FFF for positive numbers. 8000 then equals -32767, 8001=-32766, to FFFF=-1.

Back to the story. Changing worked in all of my trials, and oleBuzzard is testing it as I write this. I'm pretty confident this one will work.

[Next day...]
Sure enough, it's good! Another happy crack. My thanks to oleBuzzard for acting as a "beta tester" with his board to make sure this was truly cracked. It seemed like it was, but with random things you can never be too sure. No sense in sending out something that's going to break on people.

Phew!
This one was somewhat tough, but very interesting. I got the first (flawed) crack done in a night, but finding the stuff about the way LEA was working and everything took me another day or two. Then another two days to figure out the RegRoutineCool solution. There are some other Hermes externals I'm working on which are just hellish, but I picked up some things here that will help me with them. I don't know if I'm up to cracking them yet, but I'll sure be trying. I'd say which ones, but I hate vaporware. You'll know if I crack them, and if I don't, no one's left hanging.

So long, farewell, auf wiedersehen, goodbye...
I'd like to acknowledge (as I realize I haven't before) The Shepherd's excellent work on his "Assembly for Cracking" file, which was what sparked this series. If you're reading this and haven't already, look for that file as well. I use it whenever I have an assembly question (this is fairly frequent), and it's very rarely left me unanswered.

If you've done some Mac programming in C or Pascal, you might also want to think about picking up a copy of "Debugging Macintosh Software with Macsbug," by Konstantin Othmer and Jim Straus. It's not a Macsbug reference, but instead talks about how various Mac toolbox things and some general code is translated into assembly, and how to watch these things in Macsbug. Published in '91, so it's a little dated, but it's still great for seeing what some things in assembly mean. Apple publishes a straight Macsbug reference, but the store I went to didn't have it. I don't know how it could be incredibly better than this one, but if anyone has any comments on it, I'd like to hear them.

Also, I hear that this series is actually helpful and interesting to people. I'm thrilled to hear this. When I started, blasting out whole procedures in Resorcerer, the whole thing seemed like a novelty. Cool, but not actually so useful. It wasn't until oleBuzzard suggested I write a file about it that it even hit me other people might be interested. But interested people were, and it's even spurred at least one other person to find and publish his own crack for Dirt Bike 3.0. (Seeing this was a real head rush for me--way to go, CyboBoy.)

Any and all comments on this or other MacCracking files can go to an407599@anon.penet.fi, or Observer on Kn0wledge Phreak (719-578-8288) or (I love my life) The Keep.

Coming up in MacCracking IV: ????? If I can finish the Hermes externals I mention above, I'll do that. I might also revisit Net Watchman with my newfound Macsbug skills. But I'm always on the lookout for new things to work on, so if you have something you want cracked, or want a fresh viewpoint on something you're working on yourself, please feel free to send it to me. No promises of course, but I'm happy to take a crack (nyuk nyuk) at anything you want to throw my way.

I'll also do the same speech I did for Relax--distributing cracked shareware is just mean to its authors, real people without whom we'd have a lot less cool software. Post this file anywhere you think people will be interested, but please don't distribute a cracked Whimsy or the cracker to the masses. Thanks.

And that's it. Hope you had an OK time following this, it turned out to be much longer than the previous files. Until IV, so long!

A parting note from the "Two for the price of one" department...
I sometimes use a program called GraphicConverter (v1.7.7 /1) which uses a nag system identical to Relax: it pops up a dialog box you have to wait to dismiss. The crack for this took me well under an hour, and it's so similar to what I described in file 1 I'm not even going to dignify it with its own file. But if you want a little basic practice (or an upper after too many unsuccessful hours in Macsbug), it's a good one to check out.