asic Mac PPC Cracking Part 1

Cracking with Nosy

by Dot Com
December 12, 1997

I have read many cracking tutorials and have noticed they are pretty hard to understand. I first learned cracking thru Smeger's excellent text, "The Kool Krack Tutorial". The learning curve was pretty tough at first but I will attempt to get you rolling as quickly as possible thru examples. In this part we will use Nosy to bypass a typical serial number dialog box and other annoying alerts. There are many different approaches to bypassing a dialog box but this is one basic method I use the most often because its damn easy.

Software you will need

You will need the following software before you begin this tutorial:

```
Resorcerer 2.0
```

```
Nosy II 8/97
Macsbug 6.5.4a3c1
THINK Reference 2.0
WebCollage 1.0
```

Resorcerer is a resource editor like ResEdit. I prefer Resorcerer because of its excellent search features. We will use Resorcerer to modify code and checkout dialog boxes. Nosy is a PPC disassembler and will convert all assembly code into something we can read and understand. Macsbug is a debugger and dissassember that will allow us to see what code is executing. You will not need Macsbug in this lesson but get it anyway. THINK Reference is a handy utility that tells you all about Mac Toolbox Traps (more on this in a bit). You will also need to locate a copy of StarNine's WebCollage 1.0 from your local warez site (it may also be downloadable from http://www.starnine.com). This is the app that you are going to attempt to crack.

Basic Assembly Language

I have to admit that I know nothing about assembly language (I dont even know that the hell it is!) but I have not found it necessary to know a whole lot, so I wont bore you with all that stuff.

There are really only 4 pieces of assembly knowledge that you really need to know (at least for now) with the first being conditional branches.

Conditional branches are lines of code that compare values and go to or "branch" to another line of code. If your familiar with the old `BASIC` language its like an `IF THEN` statement. `IF` <serialnumber> = <12345> `THEN` <linenumber>. There are two types of branches we need to know which are `BNE` (branch if not equal) and `BEQ` (branch if equal). The machine language equivalent codes for these are 40 and 41. What the hell is machine language you say? Hell if I know! This will make sense later so bare with me. We use branches to force the program to do something different from what it normally does, kinda like a detour. A condtional branch statement looks like this:

```
4182 0014  bc   IF,cr0_EQ,laq_3
```

so this line of code branches if the value `cr0` is equal and goes to procedure `laq_3` (in Macsbug it will be a numeric value or line number, more on that later)

The next type of assembly we need to know is `NOP` which stands for No Operation. This is basically a line that does nothing..our macs just skip over this line. The machine language equivalent is 6000. We use this to delete lines of code. A `NOP` statement looks like this:

```
6000 0000  nop
```

A line that loads a procedure or subroutine is also a branch but I feel its a little different than a normal branch. A `BL` or "branch load" is just like a `GOSUB` in ole `BASIC` and it looks like this:

```
4BFF FFDD  bl    proc13
```

so this line of code will run all the code in `proc13`.

And last but not least we need to know `LI`.   What does `LI` do? I have no idea, but it means "Load Immediate" I believe. All I know is that they are useful for seeking out Dialog box ID numbers. A `LI`   statement looks like this:

```
38A0 06A4   li     r5,$6A4
```

Using Resorcerer

Upon running WebCollage Editor we of course get the typical register me dialog box as shown in Figure 1.

ell since we dont seem to have an authorization key handy I guess that where gonna have to bypass this dialog box somehow and get the program to load up for us.   Most (not all) dialog boxes in an application have a Dialog ID number.   We will use Resorcerer to locate this number. Load up Resorcerer and load in WebCollage Editor and select the `DLOG` Resource and we see Figure 2.

e find that Dialog number 1701 “add key dialog” is the problem. Sometimes they dont label there dialogs as did StarNine so you will have to load up each dialog and find the right one. Now we need to convert the ID number into Hex so we can locate it in Nosy. What is Hex anyway you say? Well you guessed it, I dont know. I guess its just numbers that assembly code can recognize, who knows? Anyway go to the Edit Menu and select “Value Converter” and type in 1701 in the Long field as seen in Figure 3.

ow in Hex, 1701 = 06A5. Now its time to load up Nosy and snoop around the Dialog traps and try to locate the annoying procedure that calls Dialog 1701.

Toolbox Traps

Smeger described Toolbox Traps better than I can:

The Toolbox is a set of routines that mac programmers can use to simplify common tasks, making writing code really simple 'cause you don't have to do anything. A trap is a system routine that performs some sort of action, such as drawing a menu bar or a window. Traps are stored within a program as a single instruction. When the trap is called, the program will perform the trap, then continue execution normally.

Got That? Good. For a complete listing of Toolbox Traps and what they do I use the THINK Reference app. You should be able to find it no problem. The trap calls we are interested in are any traps relating to Dialog Boxes such as `GetNewDialog`, `GetDialogItem`, etc. `GetNewDialog` seems to be the best one and I use it everytime.

Using Nosy

Nosy is a great program that will disassemble the Data Fork (where PPC code is located) into a format that we can read. First duplicate WebCollage Editor and rename the copy to just Editor (Nosy only accepts 20 character filenames) and load it into Nosy. Select the `<DF>` when Nosy asks you to select a resource. Press Continue for the Treewalk optons and let Nosy explore. Nosy will take a few minutes and dissassemble the program. The time it takes varies on the filesize. Sometimes Nosy will not be able to dissassemble part of a program and Macsbug will have to be used instead which will be covered in Part 2 of this series. In this case Nosy loads the Editor fine and we get a window displaying all the Code Blocks as in Figure 4. Keep scrolling down and you will find all the Toolbox Traps used in the application.

croll down until you find `.GetNewDialog_GL_`. Select it and press âŒ˜-R (also under the Display Menu: Show Refs to). This will show all procedures that reference the `GetNewDialog` trap. We now get the window in Figure 5.

ell we lucked out as only 3 procedures have new dialog calls. Sometimes there can be 20 or more and you will have to snoop thru them all. Anyway lets check out `proc3233`. Select it and press âŒ˜-D (also under Display: Code Blk) and we get Figure 6. Scroll down a bit until you find a `BL` for the `.GetNewDialog_GL`.

otice above the first `GetNewDialog` we have a:

```
8C8C0: 3860 06A5       li       r3,$6A5
```

Thats our man as we see Hex ID `6A5` being loaded just before the dialog call. Bypassing this routine should solve the problem right? Nope. Notice the branch at address `8C8BC: bc IF,cr0_EQ,mkg_1`. If we reroute that branch to `mkg_1` it still loads the dialog and there are no branches at the start of `proc3233` to bypass all of this. So what do we do now? We bypass this whole procedure altogether. Scroll back up to the top of `proc3233` and we see the following:

```
;-refs - proc3232  proc3250
```

This shows all the references to this procedure. Lets âŒ˜-D `proc3232` and we see that nothing is apparant but the `BL` to `proc3233`. What we want to find is a conditional branch before loading `proc3233`. Its not here so lets checkout the only ref to `proc3232` which is `proc3231`. Well no conditional branches in `proc3231` as well but an interesting `LI` line referencing `$6A4`. If we type `6A4` into the Value Converter in Resorcerer we find that its doing something with Dialog 1700 “key list dialog”, definetly something we want to avoid. The only ref to this procedure is `proc20`, lets check it out. Well, `proc20` is a big one. Do a search for `proc3231` with a âŒ˜-F. Type in `proc3231` and it we find the contents in Figure. 7.

f we keep clicking the Find button we see that this is the only occurance of `proc3231` in this procedure. Well there it is; a conditional branch above the call to `proc3231`:

```
4182 0048      1001DA8    bc       IF,cr0_EQ,lau_9
```

Notice the `NoteAlert` in Figure 7, the `LI` above it calls $14D. Go back to the Value Converter in Resorcerer and type in 14D and we get 333. Select the `ALRT` (Alert dialogs are located here) resource type and double click on 333 and we get a nice `ALRT` saying that "None of the authorization keys are valid.
Please contact StarNine for more information". Well this has got to be the right place. It looks like if we force the application to reroute to `lau_9` then we should avoid the serial number dialog box and the `ALRT`. All we have to do is change it from a BEQ to a BNE. How do we do that? Easy. First lets examine the code around the line we want to change:

```
4808 D7B5      108F508
   bl       proc3267
6000 0000                 nop
7C60 0735



              extsh.   r0,r3
4182 0048      1001DA8

  bc       IF,cr0_EQ,lau_9
3860 FFFF                 li       r3,-1
3880 004C                 li       r4,76
```

See the number 41820048? That is the assembly code for this line. What we are going to do is search the `<DF>` in Resorcerer for this line. What we find is that there are many 41820048's in the `<DF>` so we need to copy down the surrounding code so we make sure we are in the right place. The surrouding code would be `600000007C60073541820048 3860FFFF3880004C`

Modifiying Code in Resorcerer

To make the change we will use Resorcerer. In Resorcerer and select the `<DF>` resource type and do a âŒ˜-F and copy in the code with NO spaces in between the codes as in Figure 8.

elect the “Of Type `<DF>`” and “Hex” checkboxes and click Find. What we get is the window in Figure 9.

ow we are going to change 41820048 to 40820048, the exact opposite (a BEQ to a BNE).

Select the first two digits (41) of the code and type in 40.

Completing the Crack

Close out of everything and save changes and lets see what happens.   Well our crack is still not complete   as ole `ALRT` 333 still pops up.   We did, however, got rid of the serial dialog. Lets go back to Nosy and checkout all the refs to `.NoteAlert_GL_` and we find only two: `proc6` and `proc20`.   Well we already bypassed the call in `proc20` so lets checkout `proc6` and lo and behold we have a conditional branch above the `LI` call to `ALRT $14D` in Figure 10.

gain, it looks like if we change:

```
4182 0044      10008A0   bc   IF,cr0_EQ,lag_1
```

we can bypass the ALRT.   Do the same as before and copy the code around it and make the change to 4182044 to 4082004 and see what happens.

Bingo! the program loads up with no problems.   One thing you might notice is the menu command "Edit Authorization Keys" under the Edit menu.   You might want to modify the Edit menu in Resorcerer under the MENU resource type and delete it.   This will make sure that nobody can get to any annoying serial number number dialogs and have the program quit.

Adios

Hopefully this was helpful and you are well on your way to cracking your own software.   You can now apply your newly learned cracking skills to the WebCollage Assembler.

In Part 2 we will cover basic Macsbug cracking when Nosy poops out on us (a common occurance).   If you need help or have questions you can usually find me on #macfilez.

Good Luck,

Dot Com

Special thanks to sm00th who got me started in cracking and Dream for giving me inspiration to write this.