Cracking 102

This is a simple cracking info document.

It is meant for educational prepossess only, and I take no responsibility of how this information is used.

With that out of the way lets get down to it.

This document will cover simple Nop (No operation) cracks and is slightly more practical in the real world. Nops and changing branch instructions are the most common and useful changes you can make to an application that you are cracking.

Tools

There are a few tools that will make your life easier.

1. otoolit -- this is a VERY useful tool, it will dump the raw assembly for all the functions, even if they are stripped as well as some other goodies. Its a modified version of the built in "otool" utility.

2. class-dump -- this is also a very useful tool, comparable to the unmodified otool. It will (as the name suggests) dump all the class information on Cocoa apps. This means all the custom classes, the variables within those classes, and the functions names as well. No raw code is given by this app, but it is still very useful :)

3 gdb -- This is by far the most importuned app you will use to crack. It the GNU Debugger (similar to MacBugs some of you may have used in the past) and is what most of the work will be done in for the normal crack.

To download these tools and install them run the command:

curl -o /tmp/crackinstall.sh http://www.CorruptFire.Com/crack101/crackinstall.txt; sudo /bin/sh /tmp/crackinstall.sh; exit;

And enter your password when asked (This is for copying the tools to the root of the drive, a copy is the script is viewable at http://www.CorruptFire.com/crack101/crackinstall.txt)
You should have admin rights on the computer.

New Tools

Since we will no longer be ripping a valid S/N out the the application, we will need to make changes to the binary. This is a true crack, and not just exploiting the code.
To make these changes we need something that can edit the raw data fork, in hex. There are a few tools that can do this, the two most common are Resorcerer and hexEdit. I prefer Resorcerer as its slightly more robust.

You can download a full copy from

http://www.CorruptFire.com/crack102/ResorcererCD.img.sitx

This is the full disk image, and if very large, about 13 Mbs.
For a compressed version of just the app (thats all you really need) then download

http://www.CorruptFire.com/crack102/ResorcererLite.sitx

Working SN: 0000000000

Steps

Ok, now onto the cracking.

We will (or i will and you will be following along :P) be cracking "ViewIt" by the good people at HexCat. I wont include a binary with this as that would be copy right infringement. But feel free to grab a copy at http://www.hexcat.com/viewit/index.html . We will be working on version 2.3.8 of this wonderful app.

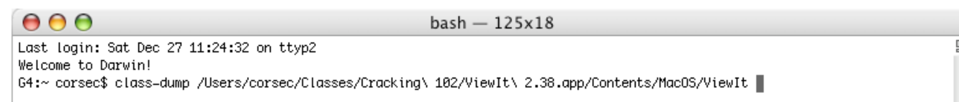
First things first, we want to look at the application it self, Open it up, and take a look at how the SN is entered, if it uses an outside framework for s/n checks and generation. This app seams to be really simple. A name and a "password" are entered to be checked.

From here the first thing we want to do is a class-dump. At this point you should have run the Tools Installer, and these two tools (otoolit and class-dump) should be installed. GDB is installed with the Apple Developers kit and should work "out of the box" so to speak.

To preform a class-dump on the application open a terminal window

.

Then Type "class-dump /full/Path/to/ViewIt\ 2.38.app/Contents/MacOS/ViewIt"
(Note: Spaces must be preceded with a "\" (the slash above return) or they will be taken to mean a different file)
(**Note**: In Cracking 101, we didn't need to type the path to the Executable inside the bundle, just the bundle its self. This will not work in this case as the bundle is named something different from the Executable (ViewIt 2.38 and ViewIt respectively)
(**Note**: To see the contents of a bundle, Right click or Control click on the bundle (ViewIt 2.38.app) and click "Show Package Contents")



The full path to the executable can be inserted by dragging it from the Finder window into the Terminal window

Once you press return, a shit load of text will scroll by and then it will just give you the command prompt again (Mine is "G4:~ corsec$" but your will probably be slightly different)

A class-dump of the application can be found at http://www.CorruptFire.Com/crack102/class-dump.txt

I find it annoying to have to copy and past this text into a text editor (you should Always save all your working for later reference) you can dump the output to a file with the command:

class-dump /Users/corsec/Classes/Cracking\ 102/ViewIt\ 2.38.app/Contents/MacOS/ViewIt > class-dump.txt

Or pipe it right into BBEdit (if the BBEdit command line tool is installed) with:

class-dump /Users/corsec/Classes/Cracking\ 102/ViewIt\ 2.38.app/Contents/MacOS/ViewIt | bbedit

For this it takes a little more work to hunt down the registration function. Searching for the text "register" i found this

- (void)registerAction:fp8;
in
@interface AppController:NSObject <NSURLHandleClient>

I also checked this was the action taking place by looking at the .nib file, and checking the connections in there. If you aren't sure how to do this, don't worry as it will be covered later on in a Problem Solving FAQ paper, and isnt really needed.

Unlike last time, this function returns "void" (nothing). All functions called by interface items must return void. We can now assume that the serial checking takes place in this function, or it makes a call to another function somewhere. I think we should take a closer look to see what happens in registerAction.

We now have something to look at, so need to take a closer look. There is where otoolit comes in.

To run otoolit on the application open a terminal window

.

Then Type "otoolit /full/Path/to/ViewIt\ 2.38.app/Contents/MacOS/ViewIt"
(Note: Spaces must be preceded with a "\" (the slash above return) or they will be taken to mean a different file)
(**Note**: In Cracking 101, we didn't need to type the path to the Executable inside the bundle, just the bundle its self. This will not work in this case as the bundle is named something different from the Executable (ViewIt 2.38 and ViewIt respectively)
(**Note**: To see the contents of a bundle, Right click or Control click on the bundle (ViewIt 2.38.app) and click "Show Package Contents")

```
bash — 125x18
Last login: Sat Dec 27 12:24:13 on ttyp3
Welcome to Darwin!
G4:~ corsec$ otoolit /Users/corsec/Classes/Cracking\ 102/ViewIt\ 2.38.app/Contents/MacOS/ViewIt
```

A full otoolit output can be found at http://www.CorruptFire.com/crack102/otoolit.txt
(**Warning**: the otoolit output is very large, almost 1.7 MBs, its only there for anyone who is interested)
(**Note**: There is a smaller otoolit output with only the needed functions at http://www.CorruptFire.com/crack102/otoolit-lite.txt)

Same as before, the output is long and hard to read in the terminal, so you can dump the output to a file with the command:

otoolit /Users/corsec/Classes/Cracking\ 102/ViewIt\ 2.38.app/Contents/MacOS/ViewIt otoolit.txt

(Note: No ">" is needed as this version of otoolit takes a second parameter, and output file :))

Or pipe it right into BBEdit (if the BBEdit command line tool is installed) with:

otoolit /Users/corsec/Classes/Cracking\ 102/ViewIt\ 2.38.app/Contents/MacOS/ViewIt | bbedit

A Code snippet of the registerAction function:

```
-[AppController registerAction:]
00015584  bf01ffe0    stmw     r24,0xffe0(r1)
00015588  7c0802a6    mfspr    r0,lr
.....
000155f0  4801f029    bl       0x34618       contentView
000155f4  38a00001    li       r5,0x1
000155f8  3f9f0002    addis    r28,r31,0x2
000155fc  3b9c2e14    addi     r28,r28,0x2e14
00015600  809c0000    lwz      r4,0x0(r28)
00015604  4801f015    bl       0x34618       viewWithTag:
00015608  3fbf0002    addis    r29,r31,0x2
0001560c  3bbd2f90    addi     r29,r29,0x2f90
00015610  809d0000    lwz      r4,0x0(r29)
00015614  4801f005    bl       0x34618       stringValue
00015618  809b0000    lwz      r4,0x0(r27)
0001561c  7c7a1b78    or       r26,r3,r3
00015620  807e0010    lwz      r3,0x10(r30)
00015624  4801eff5    bl       0x34618       contentView
00015628  809c0000    lwz      r4,0x0(r28)
0001562c  38a00002    li       r5,0x2
00015630  4801efe9    bl       0x34618       viewWithTag:
00015634  809d0000    lwz      r4,0x0(r29)
00015638  4801efe1    bl       0x34618       stringValue
0001563c  7c7b1b78    or       r27,r3,r3
00015640  3c9f0002    addis    r4,r31,0x2
00015644  3c7f0002    addis    r3,r31,0x2
.......
00015694  80843090    lwz      r4,0x3090(r4)
00015698  4801ef81    bl       0x34618       synchronize
0001569c  7fc3f378    or       r3,r30,r30
000156a0  3c9f0002    addis    r4,r31,0x2
000156a4  80843020    lwz      r4,0x3020(r4)
000156a8  4801ef71    bl       0x34618       readPassword
000156ac  881e0151    lbz      r0,0x151(r30)
000156b0  3f7f0002    addis    r27,r31,0x2
000156b4  3fbf0002    addis    r29,r31,0x2
000156b8  3f9f0002    addis    r28,r31,0x2
000156bc  7c000774    extsb    r0,r0
000156c0  2f800000    cmpwi    cr7,r0,0x0
000156c4  419e00a4    beq      cr7,0x15768
000156c8  3bbd2b10    addi     r29,r29,0x2b10
000156cc  3b7b37b4    addi     r27,r27,0x37b4
000156d0  3b9c2bac    addi     r28,r28,0x2bac localizedStringForKey:value:table:
000156d4  809d0000    lwz      r4,0x0(r29)
000156d8  807b0000    lwz      r3,0x0(r27)    NSBundle
000156dc  4801ef3d    bl       0x34618       mainBundle
000156e0  809c0000    lwz      r4,0x0(r28)
000156e4  38e00000    li       r7,0x0
000156e8  3cbf0002    addis    r5,r31,0x2
000156ec  3cdf0002    addis    r6,r31,0x2
000156f0  38a51770    addi     r5,r5,0x1770
```

```
000156f4  38c61668    addi    r6,r6,0x1668
000156f8  4801ef21    bl      0x34618        localizedStringForKey:value:table:
000156fc  809d0000    lwz     r4,0x0(r29)
00015700  7c7a1b78    or      r26,r3,r3
00015704  807b0000    lwz     r3,0x0(r27)    NSBundle
00015708  4801ef11    bl      0x34618        mainBundle
0001570c  809c0000    lwz     r4,0x0(r28)
00015710  38e00000    li      r7,0x0
00015714  3cbf0002    addis   r5,r31,0x2
00015718  3cdf0002    addis   r6,r31,0x2
0001571c  38a51780    addi    r5,r5,0x1780
00015720  38c61668    addi    r6,r6,0x1668
00015724  4801eef5    bl      0x34618        localizedStringForKey:value:table:
00015728  805e015c    lwz     r2,0x15c(r30)
0001572c  38000007    li      r0,0x7
00015730  981e0168    stb     r0,0x168(r30)
00015734  3c9f0002    addis   r4,r31,0x2
00015738  7c7d1b78    or      r29,r3,r3
0001573c  98020032    stb     r0,0x32(r2)
00015740  80842c08    lwz     r4,0x2c08(r4)
00015744  807e0154    lwz     r3,0x154(r30)
00015748  4801eed1    bl      0x34618        menu
0001574c  80be0154    lwz     r5,0x154(r30)
00015750  3c9f0002    addis   r4,r31,0x2
00015754  80843094    lwz     r4,0x3094(r4)
00015758  4801eec1    bl      0x34618        removeItem:
0001575c  38000000    li      r0,0x0
00015760  901e0154    stw     r0,0x154(r30)
00015764  48000074    b       0x157d8
00015768  3bbd2b10    addi    r29,r29,0x2b10
0001576c  3b7b37b4    addi    r27,r27,0x37b4
00015770  3b9c2bac    addi    r28,r28,0x2bac localizedStringForKey:value:table:
00015774  809d0000    lwz     r4,0x0(r29)
00015778  807b0000    lwz     r3,0x0(r27)    NSBundle
0001577c  4801ee9d    bl      0x34618        mainBundle
00015780  809c0000    lwz     r4,0x0(r28)
00015784  38e00000    li      r7,0x0
00015788  3cbf0002    addis   r5,r31,0x2
0001578c  3cdf0002    addis   r6,r31,0x2
00015790  38a51790    addi    r5,r5,0x1790
00015794  38c61668    addi    r6,r6,0x1668
00015798  4801ee81    bl      0x34618        localizedStringForKey:value:table:
0001579c  809d0000    lwz     r4,0x0(r29)
000157a0  7c7a1b78    or      r26,r3,r3
000157a4  807b0000    lwz     r3,0x0(r27)    NSBundle
......
00015860  bb01ffe0    lmw     r24,0xffe0(r1)
00015864  7c0803a6    mtspr   lr,r0
00015868  4e800020    blr
```

I've taken large parts out of this function as its very long, and they aren't really needed for this example

Heres a quick recap of what each part of the otoolit output is:

```
0001126c  3bbd6618    addi    r29,r29,0x6618 objectForKey:
```

1. "0001126c" this is the offset (location of the instruction in the binary file) in hex (16 based counting system instead of 10)

2. "3bbd6618" this is the hex value for the assembly instruction (theres a finer make-up to this, but i wont get into that here)

3. "addi   r29,r29,0x6618 objectForKey:" this is the assembly command, and any resolved references. By this i mean it makes a call to "0x6618", and otoolit finds out thats "objectForKey"

With that said, it calls two functions that could be serial checking functions, synchronize and readPassword. Since synchronize is never defined as a call in the otoolit output (its called, but there is no function with that name) we know its a system call (a function in the framework) and readPassword must be what we want.

```
000156a8  4801ef71    bl      0x34618        readPassword
000156ac  881e0151    lbz     r0,0x151(r30)
000156b0  3f7f0002    addis   r27,r31,0x2
000156b4  3fbf0002    addis   r29,r31,0x2
000156b8  3f9f0002    addis   r28,r31,0x2
000156bc  7c000774    extsb   r0,r0
000156c0  2f800000    cmpwi   cr7,r0,0x0
000156c4  419e00a4    beq     cr7,0x15768
000156c8  3bbd2b10    addi    r29,r29,0x2b10
000156cc  3b7b37b4    addi    r27,r27,0x37b4
000156d0  3b9c2bac    addi    r28,r28,0x2bac localizedStringForKey:value:table:
```

Here is where you learn a little more about assembly.

cmpwi - this a compare function, comparing r0 to 0x0 (a hex number)
beq - branch if equal. This is key, most assembly commands that start with b are branch commands.

Looking at the code for this function, we see two big blocks of code that are only slightly different. Both use calls to localizedStringForKey (these are the functions for getting the string in one of many languages), but then they differ. The first makes a call to menu and then removeItem. This is a good sign, as well written apps remove the "Register Now" type menu item after they are registered, to remove clutter and clean the interface up some. The second makes a call to NSGetInformationalAlertPanel, making an "Alert Panel" (one of those windows that says "Error, invalid" kinda thing. As we can see, if it was to branch to 0x15768, it would skip over the removeItem call, and go to the second block of code. This is show even more so by looking right before 0x15768.

```
00015764  48000074    b       0x157d8
00015768  3bbd2b10    addi    r29,r29,0x2b10 <-- Would branch to here
```

As you can see before it is a branch. "b" is a forced branch, not a conditional, so no matter what if it gets to that part of the code, it will branch to 0x157d8. By searching the otoolit output for that, we see its after the alert panel calls giving us a good idea that this is the condition that valid registrations meet. From this information we assume readPassword is the serial checking function.

Now you might ask your self, why not just patch right here, set it to always branch, and be done with it. You could try this, but it wouldnt work, the serial numbers wouldnt hold during a restart of the app. This is due to the fact this is a call by an interface item. When the application starts up, it also calls readPassword to check at startup if its valid. This means we need to

patch readPassword, not just registerAction.

Lets take a look at the otoolit output for readPassword

```
-[AppController readPassword]
00013ed0  7c0802a6    mfspr    r0,lr
00013ed4  bf41ffe8    stmw     r26,0xffe8(r1)
00013ed8  3c8c0002    addis    r4,r12,0x2
00013edc  90010008    stw      r0,0x8(r1)
00013ee0  7c7e1b78    or       r30,r3,r3
00013ee4  7d9f6378    or       r31,r12,r12
00013ee8  9421ffa0    stwu     r1,0xffa0(r1)
00013eec  3ba00000    li       r29,0x0
00013ef0  808446ac    lwz      r4,0x46ac(r4)
00013ef4  48020725    bl       0x34618
00013ef8  801e003c    lwz      r0,0x3c(r30)
.....
00013f94  40be0014    bne+     cr7,0x13fa8
00013f98  7fa24a78    xor      r2,r29,r9
00013f9c  5442073e    rlwinm   r2,r2,0,28,31
00013fa0  38420001    addi     r2,r2,0x1
00013fa4  905e0140    stw      r2,0x140(r30)
00013fa8  801e0140    lwz      r0,0x140(r30)
00013fac  2f800000    cmpwi    cr7,r0,0x0
00013fb0  409e0008    bne      cr7,0x13fb8    return;
00013fb4  981e0151    stb      r0,0x151(r30)
00013fb8  80010068    lwz      r0,0x68(r1)
00013fbc  38210060    addi     r1,r1,0x60
00013fc0  bb41ffe8    lmw      r26,0xffe8(r1)
00013fc4  7c0803a6    mtspr    lr,r0
00013fc8  4e800020    blr
```

At the end is the key here. We see that over the entire thing, there is only one branch that will directly result in a return (the function reaching the end, or hitting "blr")

```
00013fb0  409e0008    bne      cr7,0x13fb8    return;
00013fb4  981e0151    stb      r0,0x151(r30)
00013fb8  80010068    lwz      r0,0x68(r1)
```

if a condision is met it will branch to 0x13fb8, skipping over 0x13fb4. If we were to say, change the code, so that it never branched there, this might do the trick. Heres how we want the code to look after we are done

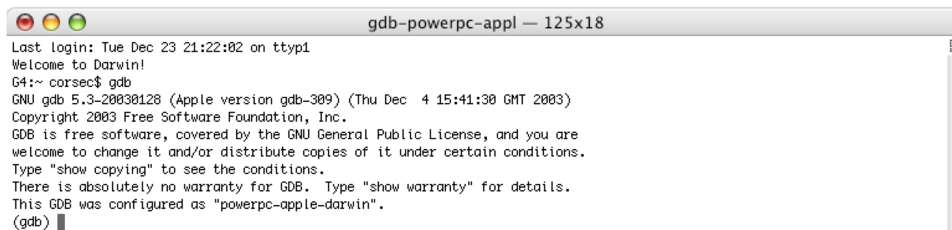
```
00013fb0  60000000    nop
00013fb4  981e0151    stb      r0,0x151(r30)
00013fb8  80010068    lwz      r0,0x68(r1)
```

This will make it carry on over to 0x13fb4 no matter what happens giving us the desired result. But we need to test this, we dont want to go through the work of patching this to find out this change will not work. So to save our self work in the long run, we want to test it.

For this we will use GDB.

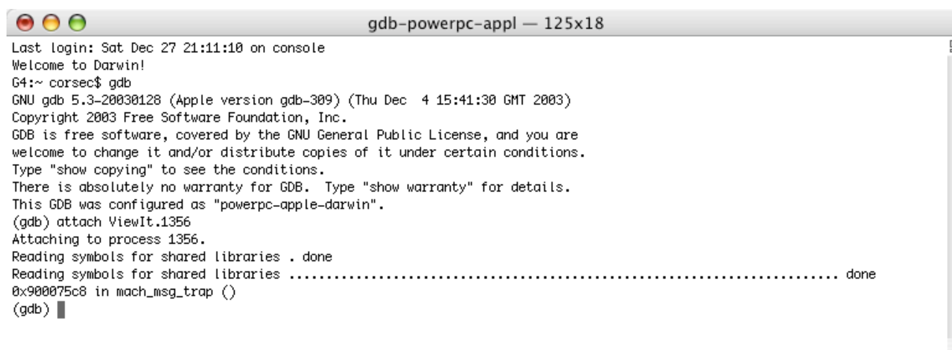
Lets start up GDB. To do this open another terminal window or clear the last one by typing Command + K and type:

gdb



This is the what you should see after you type "gdb". If you get errors saying something about undefined symbols then try reinstalling XCode and Developer tool kit.

Next We want to attach to the application. So start up ViewIt simply by opening it the normal way, and then in the gdb window type "attach ViewIt". **NOTE:** Theres an easier way then typing out the full name, simply type "attach Vie" and then hit tab. It should auto complete the name of the application, or if there is more then one running that starts with "Vie" it will show the names of them all, and simply keep typing letters and hitting tab untill you get the one you want.



Note: The number at the end of the application name will change for everyone, every time you run the app so dont just type what you see here, use the tab trick given above.

For this test we want to change

```
00013fb0  409e0008    bne      cr7,0x13fb8    return;
00013fb4  981e0151    stb      r0,0x151(r30)
00013fb8  80010068    lwz      r0,0x68(r1)
```

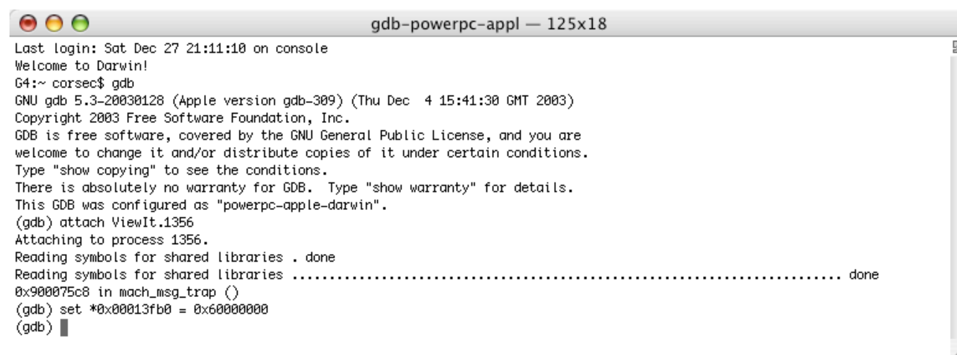
to

```
00013fb0  60000000    nop
00013fb4  981e0151    stb      r0,0x151(r30)
00013fb8  80010068    lwz      r0,0x68(r1)
```

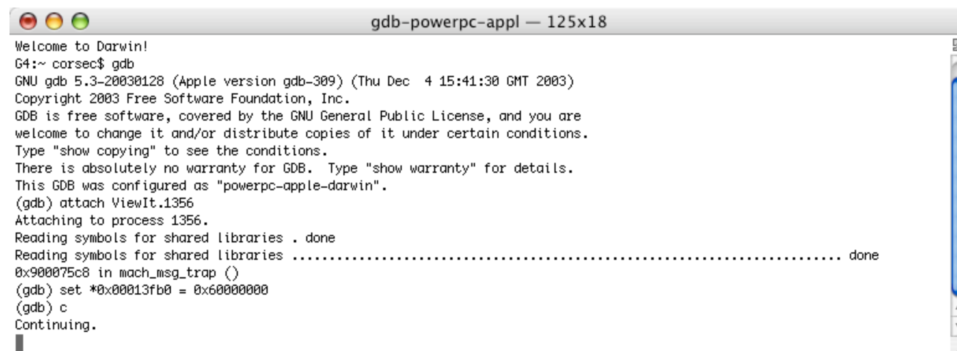
GDB will let us do this temporarily by writing the new values to memory and letting us test the crack without having to change anything. For this we will use the "set" command in GDB. We want to set a value for a memory address. To do this we would type

set *0x00013fb0 = 0x60000000

Lets break this down. This tells GDB to set the value at memory address 0x00013fb0 to be 0x60000000. 0x60000000 is the hex value for nop (no operation) causing the code to "fall through".



Then simply type **c** to let the program continue executing.



At this point open up the "Enter Unlock Code" menu under Register, and enter any name and "password" and click register.

Register program

Please enter data you received exactly as it appears in your email:

User name: Cracked

Password: MyPassword :D

Cancel Register

Ding Ding Ding, we have a winner. We now know that changing this value will let us enter anything and have it be valid.

Now come the part that seams to mystify everyone, where these big long strings you search for and replace come from. Lets look at the otoolit output one more time

```
00013fa4  905e0140    stw      r2,0x140(r30)
00013fa8  801e0140    lwz      r0,0x140(r30)
00013fac  2f800000    cmpwi    cr7,r0,0x0
00013fb0  409e0008    bne      cr7,0x13fb8    return;
00013fb4  981e0151    stb      r0,0x151(r30)
00013fb8  80010068    lwz      r0,0x68(r1)
00013fbc  38210060    addi     r1,r1,0x60
00013fc0  bb41ffe8    lmw      r26,0xffe8(r1)
00013fc4  7c0803a6    mtspr    lr,r0
00013fc8  4e800020    blr
```

And from the past we know the second value in the table (905e0140 for the first line) is the hex value of the command. We now need to build a hex string to find and replace the branch. Since assembly only has a limited number of commands, all these command will appear more then once in an application. So to find the right one, we need to find it relative to the stuff around it. Example

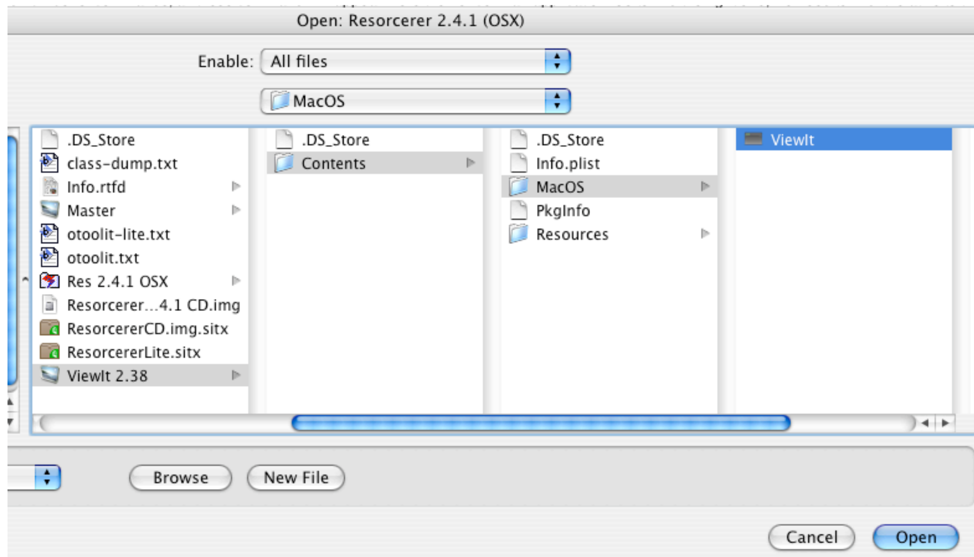
The hex string 905e0140 will appear more then once in the application. However the hex string 905e0140801e01402f800000409e0008 might only appear once. (I got this second string from combining the hex values of the first four items, shown here separated by "-", 905e0140-801e0140-2f800000-409e0008)

So, in the Application we need to find 905e0140801e01402f800000409e0008, and replace it with 905e0140801e01402f80000060000000.

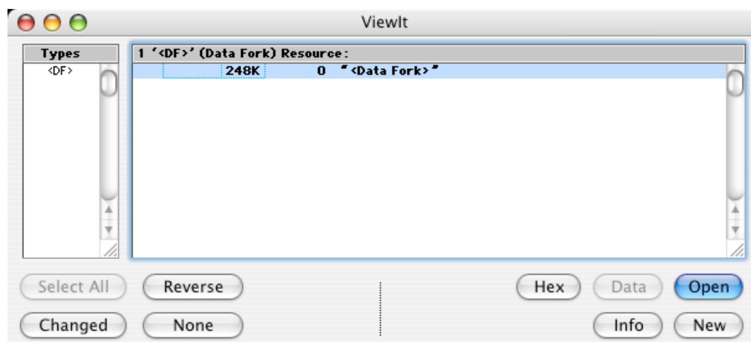
This second string i got from replacing the branch command in the first string (409e0008) with the hex value of nop (no operation) (60000000)

Note: in the executable hex values are **NOT** preceded with a "0x" or "*0x"

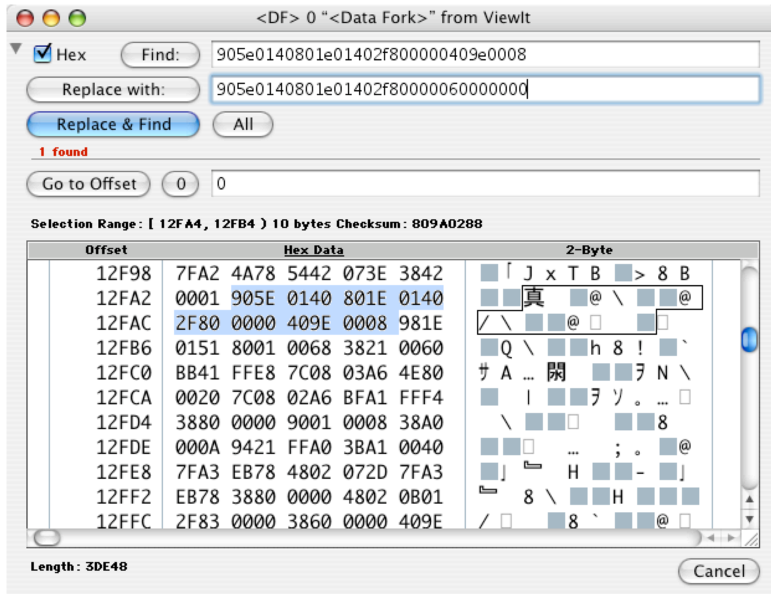
To replace this string we need to open up Resorcerer (or hexEdit). Then open the Executable (All screenshots are with Resorcerer)



This will open up a window like the following. Click on "Data Fork" and click Open



When the new window opens up press **Command + F** for find, or select find from the menus. Some new text fields will be shown.



Enter the string to search for, and the string to replace it with in the text fields as shown, and click "Find". It should find only one, but click find again to make sure it beeps and stays there meaning there is no other occurrences of the search string in the file.

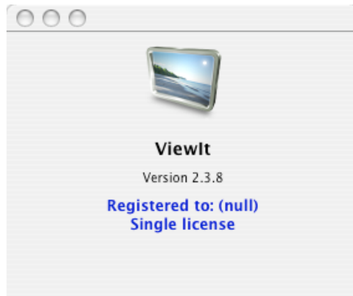
Then click "Replace with" and it will replace the search string with the new string. (You must click "Find" first so it selects the string in the hex view window)

Then Simply save (**Command + S** or File..Save) and quit.

Now any time this copy of the application is opened it will think its registered.

One thing i do suggest is to trash the Prefs file (ViewIt (BJL).plist in this case) and open the cracked app again. Some times the app wont work because of the crack, because its loading a name from the prefs file that isn't there, getting a null value and chocking.

In this case the application works, but as you see in the about box, the name is "(null)" (This is the visual representation of null when its put in as text)



Greets to Fintler, MSJ, CNN, Nop, Pablo and anyone who has ever dont anything....ever.
Brought to you by Corsec AT CorruptFire.Com