Cracking 101

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.

There are a few ways to crack apps. One of them we will be doing in this app, but other should be covered in the future. This process can be used on some shareware apps, but most shareware developers are smarter then this one, and dont code a serial generating function right into the app. This is rare, and for any of you developers out there, DONT DO THIS! Its VERY VERY BAD!

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.

Steps

Ok, now onto the cracking.

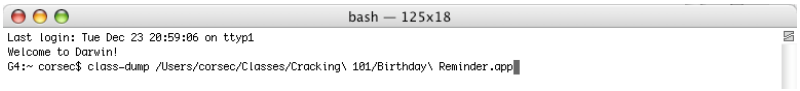
We will (or i will and you will be following along :P) be cracking "Birthday Reminder" by Michel Dalal. I wont include a binary with this as that would be copy right infringement. But feel free to grab a copy at http://www.micheldalal.com/sw/macosx/br/ . We will be working on version 1.0.3 of this wonderful 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 s/n field are all thats there. Heres a good hint, if theres a name and s/n field, most times the s/n is generated based on the name. If there is no name field chances are that there is one or two s/n's that are hard coded into the app and it just checks agents those (they are easier to crack then this).

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 Terminal.

Then Type "class-dump /full/Path/to/Birthday\ Reminder.app"
(Note: Spaces must be preceded with a "\" (the slash above return) or they will be taken to mean a different file)



The full path to the application 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/crack101/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\ 101/Birthday\ Reminder.app > class-dump.txt

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

class-dump /Users/corsec/Classes/Cracking\ 101/Birthday\ Reminder.app | bbedit

Looking at the output from this tool, something should catch your eye right away:

```
@interface LicenseManager:NSObject
{
    char licensed;
    NSString *licensee;
    NSString *realKey;
}

- (void)resetLicensedFlag;
- (void)saveValue:fp8 forKey:fp12;
- licensee;
- (void)setLicensee:fp8;
- realKey;
- (void)setRealKey:fp8;
- (char)licensed;
- getLicenseKeyForName:fp8;
- (void)setLicensee:fp8 realKey:fp12;
- init;
- (void)dealloc;

@end
```

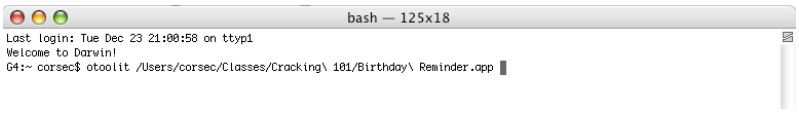
This little extract from the class-dump shows they have (its get kinda techincal here, if you dont understand, dont worry and just keep reading) sub-classed NSObject into LicenseManager, and has a function getLicenseKeyForName with one parameter. (Note: fp8 represents NSStrings) We see theres a function called "getLicenseKeyForName", Humm, i wonder what that could do ;)

We now have the function we should be looking at, and with a little guess, we assume it returns an NSString (normal string object in Cocoa)

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 Terminal.

Then Type "otoolit /full/Path/to/Birthday\ Reminder.app"
(Note: Spaces must be preceded with a "\" (the slash above return) or they will be taken to mean a different file)



A full otoolit output can be found at http://www.CorruptFire.com/crack101/otoolit.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\ 101/Birthday\ Reminder.app 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\ 101/Birthday\ Reminder.app | bbedit

This output might look a little intimidating at first, bust most of if you dont care about. Heres a code snippet:

```
-[LicenseManager getLicenseKeyForName:]
00011244  7c0802a6    mfspr     r0,lr
00011248  bf41ffe8    stmw      r26,0xffe8(r1)
0001124c  90010008    stw       r0,0x8(r1)
00011250  3c800001    lis       r4,0x1
00011254  9421fea0    stwu      r1,0xfea0(r1)
00011258  3c600001    lis       r3,0x1
0001125c  80846610    lwz       r4,0x6610(r4)
00011260  7cba2b78    or        r26,r5,r5
00011264  80636a1c    lwz       r3,0x6a1c(r3)  NSBundle
00011268  3fa00001    lis       r29,0x1
0001126c  3bbd6618    addi      r29,r29,0x6618 objectForKey:
00011270  480030a1    bl        0x14310        mainBundle
00011274  3c800001    lis       r4,0x1
00011278  80846614    lwz       r4,0x6614(r4)
..... A bunch of stuff here.....
0001130c  4bfffbdd    bl        0x10ee8
00011310  3c800001    lis       r4,0x1
00011314  7c651b78    or        r5,r3,r3
00011318  3c600001    lis       r3,0x1
0001131c  80636a38    lwz       r3,0x6a38(r3)  NSString
00011320  80846a08    lwz       r4,0x6a08(r4)
00011324  48002fed    bl        0x14310        stringWithCString:
00011328  80010168    lwz       r0,0x168(r1)
0001132c  38210160    addi      r1,r1,0x160
00011330  7c0803a6    mtspr     lr,r0
00011334  bb41ffe8    lmw       r26,0xffe8(r1)
00011338  4e800020    blr
```

This may look strange, but its not really. lets take one line and dissect it

```
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"

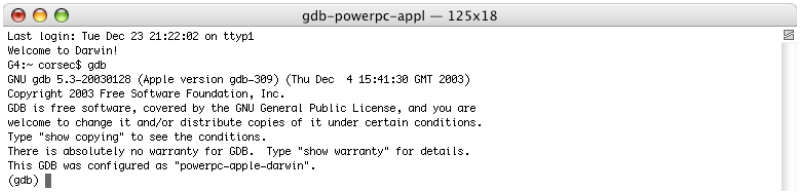
But for right now all we need is the offset of the "blr" command of "getLicenseKeyForName". We are going to assume that it returns a string (this is backed up because just before the end it makes a call to "stringWithCString", and we know that this function call (part of the built in NSString class) returns an NSString made from a cstring, as the name suggests) so we want to read the memory well the application is running, more accurately at the end of this function after all the S/N generation work is done.

For this we will use our final and most valuable tool; GDB.

GDB is a debugger, meaning it can "attach" (bind or fuse its self with an application) to an application, and watch what it is doing to memory, and also stop the code at any given point. This is what we want to do.

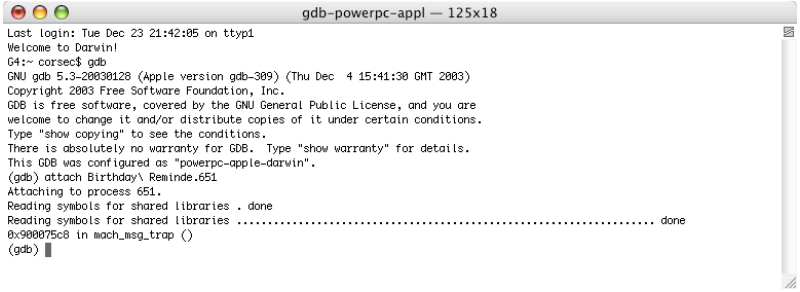
Lets start up GDB. To do this open another terminal window or clear the last one by typing Command + K and type:

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 Birthday Reminder simply by opening it the normal way, and then in the gdb window type "attach Birthday\ Reminder". **NOTE:** Theres an easier way then typing out the full name, simply type "attach Bir" and then hit tab. It should auto complete the name of the application, or if there is more then one running that starts with "Bir" 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.

Now we want to set whats called a "BreakPoint". This is a place in the code we want the application to stop, and give us a chance to do something. A breakpoint can be set a number of ways but this time we know exactly where we want the breakpoint to be.

00011338 4e800020 blr

(Taken from the otoolit output, its the end of the "getLicenseKeyForName" function.)

As i said above, the first number is the offset in hex. So we will type:

b *0x00011338

Lets break this down, first "b". This is the shortcut for breakpoint, but you can type the entire thing out of you would like :P
second the number. We tell gdb that its a offset by putting a "*" (asterisk) in front of it, and tell gdb its a hex value, by putting "0x" in front of the number.

```
0x900075c8 in mach_msg_trap ()
(gdb) b *0x00011338
Breakpoint 1 at 0x11338
(gdb)
```

We have now told gdb to stop the code at the end of the "getLicenseKeyForName" function.

We now want to tell gdb to let the code continue to run. To do this simply type "c", short form for continue.

```
(gdb) b *0x00011338
Breakpoint 1 at 0x11338
(gdb) c
Continuing.
```

The application will now respond properly, Open up the Enter License Key panel, and enter some name. The second you finish editing the name box (you click off it or press tab) it will just lock up and the spinning beach-ball will show up, In the terminal in gdb it will say

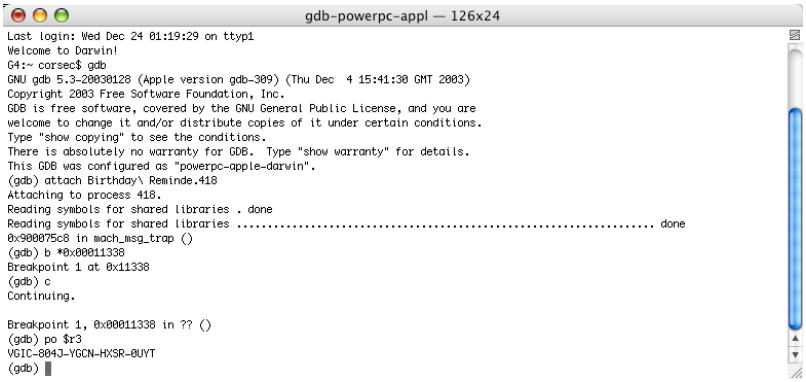
Breakpoint 1, 0x00011338 in ?? ()
(gdb)

This means you have reached the break point, and the code is holding. At this point the little things you pick up becomes importuned to the process. Im going to tell you something that you cant read in any book, but is fundamental to cracking. r3 is Whats returned. Let me explain, assembly has 32 internal variables, r0-31. When a function ends, whatever is in r3 is returned. This is importend because most functions returning 0, 1, or some other value is critical to the serial checking, and should it fail, ...or say return 1 all the time then the serial checking would become pointless :)

In this case, the function will be returning a S/N to us. For this all it needs is a simple peek of the value its going to be returned. This is done with the "print out" command as follows:

(gdb) po $r3
VGIC-804J-YGCN-HXSR-0UYT
(gdb)

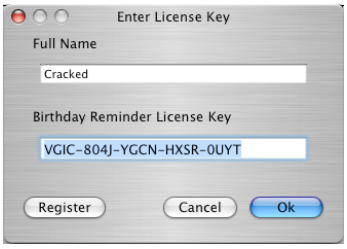
For you to use a variables value, you must put a "$" (dollar sign) in-front of it, much like the 0x in-front of hex numbers.



As you can see, out pops a valid S/N :D

Simple use this in combination with the name you entered, and its valid.

Heres a free one:



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