

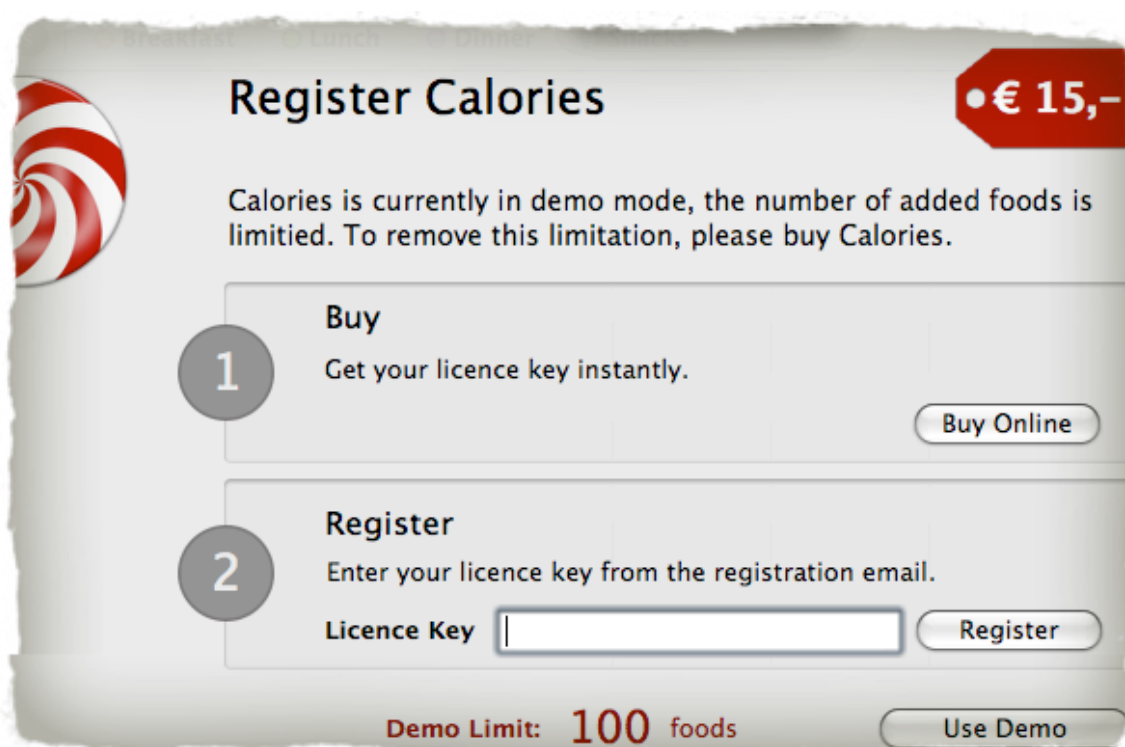
Get krackin' part 3 - Attack, analyze and reverse

Tools used: gdb, otx

In this tutorial we will be looking at an application called Calories. It uses a very simple scheme for it's serials but it should give us some basic understanding of reading, understanding and reversing a Cocoa application.

Basic overview of our target

When we launch our target app we get presented with the above sheet, asking us to enter a serial. We are also informed about the demo limit, which is a good thing if we are making a krack for this app and want to make sure the krack works. But in this tutorial we will be focusing on how to create valid serials for the app!



Finding what to attack

I often get asked how I find the right place to attack in an app. In one of the previous tutorials I described three methods of doing this:

1. Looking up actions in .nib-files.
2. Searching otx output for obvious things like "license" and hoping we find the right place
3. Using gdb

Nowadays (on Leopard) there is a problem with the first method since many .nib-files are compiled and we are unable to open them. And naturally, the second method is also not always an ideal one since searching thru 3713712 lines of code in your otx output to find

the right spot can be a tedious process that sometimes can take just a few seconds and sometimes it can take half an hour. So my advice is to use the third method and this also is the one we will be using for our target app.

Start by opening up the terminal and attach gdb to our app. Enter some random characters (in this tutorial I'll be using `123456789`). Switch back to the the terminal and press ctrl-C to interrupt the app. Now in the (gdb) prompt enter `b stringValue`

```
(gdb) b stringValue
[0] cancel
[1] all

Non-debugging symbols:
[2]  -[DOMXPathResult stringValue]
[3]  -[NSActionCell stringValue]
[4]  -[NSAppleEventDescriptor stringValue]
[5]  -[NSButtonCell stringValue]
[6]  -[NSCell stringValue]
[7]  -[NSControl stringValue]
[8]  -[NSLevelIndicatorCell stringValue]
[9]  -[NSNumber stringValue]
[10] -[NSPanelController stringValue]
[11] -[NSSliderCell stringValue]
[12] -[NSStepperCell stringValue]
[13] -[NSXMLDocument stringValue]
[14] -[NSXMLElement stringValue]
[15] -[NSXMLFidelityNode stringValue]
[16] -[NSXMLNamedFidelityNode stringValue]
[17] -[NSXMLNode stringValue]
[18] -[_NSDatePickerCellSubfield stringValue]
```

Whoah! A lot of choices there. So how do we know which breakpoint the app will hit when a stringValue gets taken from the textfield? We could find out by looking at the Cocoa documentation for textfields, or NSTextField as they are called in the Cocoa frameworks.

Inherits from **NSControl : NSView : NSResponder : NSObject**

As we can see NSTextField inherits from NSControl and one of the suggested stringValue breakpoints was `-[NSControl stringValue]`. So this is where we will set our breakpoint!

Set the breakpoint and continue execution of the app. Switch back to the app and press the “Register” button. The app will freeze because our breakpoint was hit. Back in gdb we will do a backtrace and find out what has been called before our `-[NSControl stringValue]` breakpoint was hit.

```

Breakpoint 1, 0x91209f9c in -[NSControl stringValue] ()
(gdb) bt
#0 0x91209f9c in -[NSControl stringValue] ()
#1 0x00007b9c in -[CLAppController registerApplication:] ()
#2 0x91220eb8 in -[NSApplication sendAction:to:from:] ()
#3 0x91220dec in -[NSControl sendAction:to:] ()
#4 0x91220304 in -[NSCell trackMouse:inRect:ofView:untilMouseUp:] ()
#5 0x9121fc3c in -[NSButtonCell trackMouse:inRect:ofView:untilMouseUp:] ()
#6 0x9121f578 in -[NSControl mouseDown:] ()
#7 0x9121de3c in -[NSWindow sendEvent:] ()
#8 0x911f11e0 in -[NSApplication sendEvent:] ()
#9 0x9115e4b8 in -[NSApplication run] ()
#10 0x9112ee94 in NSApplicationMain ()
#11 0x000020b4 in main ()

```

Aha! looks like `-[CLAppController registerApplication:]` was called before our breakpoint. This is a function worth examining!

We could start by deleting our breakpoint since we won't be needing it anymore. In gdb enter: `delete 1` (1 is for the breakpoint number, which in our case was 1. If we want to delete all of our breakpoints we just enter: `delete`)

Next, we will set a new breakpoint at `-[CLAppController registerApplication:]` and repeat the same procedure again (enter our license key in the textfield, press the "register" button and watch the app freeze).

Our new breakpoint was hit and this time we will use the `disas` command in gdb to disassemble this function to see if we find anything out of interest.

```

Breakpoint 2, 0x00007b78 in -[CLAppController registerApplication:] ()
(gdb) disas
Dump of assembler code for function -[CLAppController registerApplication:]:
0x00007b64 <-[CLAppController registerApplication:]+0>: mflr    r0
0x00007b68 <-[CLAppController registerApplication:]+4>: stw     r0,8(r1)
0x00007b6c <-[CLAppController registerApplication:]+8>: stwu    r1,-80(r1)
0x00007b70 <-[CLAppController registerApplication:]+12>: stw     r29,76(r1)
0x00007b74 <-[CLAppController registerApplication:]+16>: stw     r30,72(r1)
0x00007b78 <-[CLAppController registerApplication:]+20>: mr      r30,r3
0x00007b7c <-[CLAppController registerApplication:]+24>: lis     r2,4
0x00007b80 <-[CLAppController registerApplication:]+28>: lwz     r5,28(r30)
0x00007b84 <-[CLAppController registerApplication:]+32>: lwz     r4,-23856(r2)
0x00007b88 <-[CLAppController registerApplication:]+36>: lis     r29,4
0x00007b8c <-[CLAppController registerApplication:]+40>: bla     0xfffff00
0x00007b90 <-[CLAppController registerApplication:]+44>: lwz     r4,-24136(r29)
0x00007b94 <-[CLAppController registerApplication:]+48>: lwz     r3,104(r30)
0x00007b98 <-[CLAppController registerApplication:]+52>: bla     0xfffff00
0x00007b9c <-[CLAppController registerApplication:]+56>: mr      r29,r3
0x00007ba0 <-[CLAppController registerApplication:]+60>: bl      0x1cf88
<CLValidateLicenceKey>
0x00007ba4 <-[CLAppController registerApplication:]+64>: clrlwi  r2,r3,24
....OUTPUT REMOVED....

```

```

Breakpoint 2, 0x000070d7 in -[CLAppController registerApplication:] ()
(gdb) disas
Dump of assembler code for function -[CLAppController registerApplication:]:
0x000070d2 <-[CLAppController registerApplication:]+0>:  push  %ebp
0x000070d3 <-[CLAppController registerApplication:]+1>:  mov   %esp,%ebp
0x000070d5 <-[CLAppController registerApplication:]+3>:  push  %edi
0x000070d6 <-[CLAppController registerApplication:]+4>:  push  %esi
0x000070d7 <-[CLAppController registerApplication:]+5>:  sub   $0x30,%esp
0x000070da <-[CLAppController registerApplication:]+8>:  mov   0x8(%ebp),%esi
0x000070dd <-[CLAppController registerApplication:]+11>: mov   0x1c(%esi),%eax
0x000070e0 <-[CLAppController registerApplication:]+14>: mov   0x372d0,%ecx
0x000070e6 <-[CLAppController registerApplication:]+20>: mov   %eax,0x8(%esp)
0x000070ea <-[CLAppController registerApplication:]+24>: mov   %ecx,0x4(%esp)
0x000070ee <-[CLAppController registerApplication:]+28>: mov   %esi,(%esp)
0x000070f1 <-[CLAppController registerApplication:]+31>: call  0x3d12f
<dyld_stub_objc_msgSend>
0x000070f6 <-[CLAppController registerApplication:]+36>: mov   0x68(%esi),%eax
0x000070f9 <-[CLAppController registerApplication:]+39>: mov   0x371b8,%ecx
0x000070ff <-[CLAppController registerApplication:]+45>: mov   %ecx,0x4(%esp)
0x00007103 <-[CLAppController registerApplication:]+49>: mov   %eax,(%esp)
0x00007106 <-[CLAppController registerApplication:]+52>: call  0x3d12f
<dyld_stub_objc_msgSend>
0x0000710b <-[CLAppController registerApplication:]+57>: mov   %eax,%edi
0x0000710d <-[CLAppController registerApplication:]+59>: mov   %edi,(%esp)
0x00007110 <-[CLAppController registerApplication:]+62>: call  0x1ae44
<CLValidateLicenceKey>
...OUTPUT REMOVED...

```

The two above gdb outputs shows part of the gdb output on both PPC and Intel sides. What is interesting here, PPC or Intel doesn't matter, is looking at what calls (to other functions) are being done in this function. On PPC these are *bl* and *bla*. On Intel they are *call*. Now there are of course more than just these calls in our output that might be useful to read and understand as a kracker (I might write a basic assembler tutorial later..)

What is interesting in `-[CLAppController registerApplication:]` is that `CLValidateLicenceKey` that is being called. This looks like the actual place where the license key is checked and this is where we will have a closer look.

Analyzing CLValidateLicenceKey

Now that we have found a function that the license key appears to be validated in we can switch to otx and disassemble the app and have a closer look at what goes on in `CLValidateLicence`.

```

+52 0001cfbc 3c400004 lis r2,0x4
+56 0001cfc0 3c600004 lis r3,0x4
+60 0001cfc4 8083a6fc lwz r4,0xa6fc(r3)
stringWithCapacity:
+64 0001cfc8 8062ae44 lwz r3,0xae44(r2)
NSMutableString
+68 0001cfcc 38a003e8 li r5,0x3e8
+72 0001cfd0 4bfeff03 bla 0xffffeff00 +
[NSMutableString stringWithCapacity:]
+76 0001cfd4 7c7d1b78 or r29,r3,r3
+80 0001cfd8 3c400004 lis r2,0x4
+84 0001cfdc 3c600004 lis r3,0x4
+88 0001cfe0 8083a57c lwz r4,0xa57c(r3)
uppercaseString
+92 0001cfe4 8382a6f4 lwz r28,0xa6f4(r2)
appendString:
+96 0001cfe8 7fc3f378 or r3,r30,r30
+100 0001cfec 4bfeff03 bla 0xffffeff00 -[r3
uppercaseString]
+104 0001cff0 7c651b78 or r5,r3,r3
+108 0001cff4 7fa3eb78 or r3,r29,r29
+112 0001cff8 7f84e378 or r4,r28,r28
+116 0001cffc 4bfeff03 bla 0xffffeff00 -[r3
appendString:]
+120 0001d000 3c400004 lis r2,0x4
+124 0001d004 3fc00004 lis r30,0x4
+128 0001d008 809ea0a8 lwz r4,0xa0a8(r30) length
+132 0001d00c 8382aa54 lwz r28,0xaa54(r2)
replaceOccurrencesOfString:withString:options:range:
+136 0001d010 7fa3eb78 or r3,r29,r29
+140 0001d014 4bfeff03 bla 0xffffeff00 -[r3
length]
+144 0001d018 3b600000 li r27,0x0
+148 0001d01c 93610048 stw r27,0x48(r1)
+152 0001d020 9061004c stw r3,0x4c(r1)
+156 0001d024 90610030 stw r3,0x30(r1)
+160 0001d028 80410048 lwz r2,0x48(r1)
+164 0001d02c 3c600004 lis r3,0x4
+168 0001d030 9041002c stw r2,0x2c(r1)
+172 0001d034 3c400004 lis r2,0x4
+176 0001d038 8121004c lwz r9,0x4c(r1)
+180 0001d03c 81010048 lwz r8,0x48(r1)
+184 0001d040 3b400001 li r26,0x1
+188 0001d044 38a38fb0 addi r5,r3,0x8fb0 H
+192 0001d048 38c28fc0 addi r6,r2,0x8fc0 0
+196 0001d04c 7fa3eb78 or r3,r29,r29
+200 0001d050 7f84e378 or r4,r28,r28
+204 0001d054 7f47d378 or r7,r26,r26
+208 0001d058 4bfeff03 bla 0xffffeff00 -[r3
replaceOccurrencesOfString:withString:options:range:]
+212 0001d05c 93410044 stw r26,0x44(r1)
+216 0001d060 93610040 stw r27,0x40(r1)
+220 0001d064 809ea0a8 lwz r4,0xa0a8(r30) length
+224 0001d068 7fa3eb78 or r3,r29,r29
+228 0001d06c 4bfeff03 bla 0xffffeff00 -[r3
length]
+232 0001d070 28030000 cmplwi r3,0x0
+236 0001d074 418200b0 beq 0x1d124

```

```

+20 0001ae58 a1fc760300 movl
0x000376fc,%eax stringWithCapacity:
+25 0001ae5d 8b0d447e0300 movl
0x00037e44,%ecx NSMutableString
+31 0001ae63 89442404 movl %eax,
0x04(%esp)
+35 0001ae67 890c24 movl %ecx,
(%esp)
+38 0001ae6a c7442408e8030000 movl
$0x000003e8,0x08(%esp)
+46 0001ae72 e8b8220200 calll
0x0003d12f +[NSMutableString
stringWithCapacity:]
+51 0001ae77 89c7 movl %eax,
%edi
+53 0001ae79 a17c750300 movl
0x0003757c,%eax uppercaseString
+58 0001ae7e 89442404 movl %eax,
0x04(%esp)
+62 0001ae82 893424 movl %esi,
(%esp)
+65 0001ae85 e8a5220200 calll
0x0003d12f -[(%esp,1) uppercaseString]
+70 0001ae8a 8b0df4760300 movl
0x000376f4,%ecx appendString:
+76 0001ae90 894c2404 movl %ecx,
0x04(%esp)
+80 0001ae94 89442408 movl %eax,
0x08(%esp)
+84 0001ae98 893c24 movl %edi,
(%esp)
+87 0001ae9b e88f220200 calll
0x0003d12f -[(%esp,1) appendString:]
+92 0001aea0 a1a8700300 movl
0x000370a8,%eax length
+97 0001aea5 89442404 movl %eax,
0x04(%esp)
+101 0001aea9 893c24 movl %edi,
(%esp)
+104 0001aeac e87e220200 calll
0x0003d12f -[(%esp,1) length]
+109 0001aeb1 8b0d547a0300 movl
0x00037a54,%ecx
replaceOccurrencesOfString:withString:options:range:
+115 0001aeb7 894c2404 movl %ecx,
0x04(%esp)
+119 0001aebb 89442418 movl %eax,
0x18(%esp)
+123 0001aebf c744241400000000 movl
$0x00000000,0x14(%esp)
+131 0001aec7 c744241001000000 movl
$0x00000001,0x10(%esp)
+139 0001aecf c744240c88670300 movl
$0x00036788,0x0c(%esp) 0
+147 0001aed7 c744240898670300 movl
$0x00036798,0x08(%esp) H
+155 0001aedef 893c24 movl %edi,
(%esp)
+158 0001aee2 e848220200 calll
0x0003d12f -[(%esp,1)
replaceOccurrencesOfString:withString:options:range:]
+163 0001aee7 a1a8700300 movl
0x000370a8,%eax length
+168 0001aeeb 89442404 movl %eax,
0x04(%esp)
+172 0001aef0 893c24 movl %edi,
(%esp)
+175 0001aef3 e837220200 calll
0x0003d12f -[(%esp,1) length]
+180 0001aef8 85c0 testl %eax,
%eax
+182 0001aefa 7474 je 0x0001af70

```

Alright, so the above output is the first part (both PPC and Intel) of CLValidateLicence that we will examine to understand what is going on.

Once again have a look at what the calls to other functions are being made:

- [NSMutableString stringWithCapacity:]
- [uppercaseString:]
- [appendString:]
- [length:]
- [replaceOccurrencesOfString:withString:options:range:]

So what is happening here?

- We create a new NSMutableString - [NSMutableString stringWithCapacity:]
- We make it uppercase - [uppercaseString:]
- We append a string to it (our entered license) - [appendString:]
- We get the length of the string - [length:]
- Last one is a bit tricky...we replace every occurrence of H in our string with 0 (zero) - [replaceOccurrencesOfString:withString:options:range:] (I have no idea why this is though :S)

Let's continue with the next part:

```
+240 0001d078 3c400004 lis      r2,0x4
+244 0001d07c 8082a580 lwz      r4,0xa580(r2)
substringWithRange:
+248 0001d080 80410044 lwz      r2,0x44(r1)
+252 0001d084 80610040 lwz      r3,0x40(r1)
+256 0001d088 90410024 stw      r2,0x24(r1)
+260 0001d08c 90610020 stw      r3,0x20(r1)
+264 0001d090 80c10044 lwz      r6,0x44(r1)
+268 0001d094 80a10040 lwz      r5,0x40(r1)
+272 0001d098 7fa3eb78 or       r3,r29,r29
+276 0001d09c 4bfeff03 bla      0xffffff00      -[r3
substringWithRange:]
+280 0001d0a0 7c661b78 or       r6,r3,r3
+284 0001d0a4 3c400004 lis      r2,0x4
+288 0001d0a8 3c600004 lis      r3,0x4
+292 0001d0ac 80a2a704 lwz      r5,0xa704(r2)
rangeOfString:
+296 0001d0b0 38838fd0 addi     r4,r3,0x8fd0
0123456789ABCDEF
+300 0001d0b4 38610038 addi     r3,r1,0x38
+304 0001d0b8 4800a2fd bl       0x273b4      -
(struct)[r4 rangeOfString:]
+308 0001d0bc 80410038 lwz      r2,0x38(r1)
+312 0001d0c0 6c427fff xoris    r2,r2,0x7fff
+316 0001d0c4 2802ffff cmplwi  r2,0xffff
+320 0001d0c8 40820038 bne      0x1d100
```

```
+186 0001aefe a180750300          movl      0x00037580,%eax      substringWithRange:
+191 0001af03 89442404          movl      %eax,
0x04(%esp)
+195 0001af07 c744240c01000000    movl      $0x00000001,0x0c(%esp)
+203 0001af0f 89742408          movl      %esi,
0x08(%esp)
+207 0001af13 893c24          movl      %edi,
(%esp)
+210 0001af16 e814220200        calll    0x0003d12f      -[(%esp,1)
substringWithRange:]
+215 0001af1b 8b0d04770300      movl      0x00037704,%ecx      rangeOfString:
+221 0001af21 894c2404          movl      %ecx,
0x04(%esp)
+225 0001af25 89442408          movl      %eax,
0x08(%esp)
+229 0001af29 c70424a8670300    movl      $0x000367a8,(%esp)      0123456789ABCDEF
+236 0001af30 e8fa210200        calll    0x0003d12f      -[(%esp,1) rangeOfString:]
+241 0001af35 3dffffff7f        cmpl      $0xffffffff,%eax
+246 0001af3a 751e          jne      0x0001af5a
```

This basically checks if our entered license key contains the characters 0-9 and A-F (ie, checks that it only contains hexadecimal characters)

```

+324 0001d0cc 3c400004 lis r2,0x4
+328 0001d0d0 8082a6e8 lwz r4,0xa6e8(r2)
deleteCharactersInRange:
+332 0001d0d4 80410044 lwz r2,0x44(r1)
+336 0001d0d8 80610040 lwz r3,0x40(r1)
+340 0001d0dc 90410024 stw r2,0x24(r1)
+344 0001d0e0 90610020 stw r3,0x20(r1)
+348 0001d0e4 80c10044 lwz r6,0x44(r1)
+352 0001d0e8 80a10040 lwz r5,0x40(r1)
+356 0001d0ec 7fa3eb78 or r3,r29,r29
+360 0001d0f0 4bfeff03 bla 0xffffeff00 -[r3
deleteCharactersInRange:]
+364 0001d0f4 80410040 lwz r2,0x40(r1)
+368 0001d0f8 3842ffff addi r2,r2,0xffff
+372 0001d0fc 90410040 stw r2,0x40(r1)

+376 0001d100 80410040 lwz r2,0x40(r1)
+380 0001d104 3c600004 lis r3,0x4
+384 0001d108 3bc20001 addi r30,r2,0x1
+388 0001d10c 93c10040 stw r30,0x40(r1)
+392 0001d110 8083a0a8 lwz r4,0xa0a8(r3)
length
+396 0001d114 7fa3eb78 or r3,r29,r29
+400 0001d118 4bfeff03 bla 0xffffeff00 -[r3
length]
+404 0001d11c 7c1e1840 cmplw r30,r3
+408 0001d120 4180ff58 blt 0x1d078

```

```

+248 0001af3c a1e8760300 movl
0x000376e8,%eax deleteCharactersInRange:
+253 0001af41 89442404 movl %eax,
0x04(%esp)
+257 0001af45 c744240c01000000 movl
$0x00000001,0x0c(%esp)
+265 0001af4d 89742408 movl %esi,
0x08(%esp)
+269 0001af51 893c24 movl %edi,
(%esp)
+272 0001af54 e8d6210200 calll
0x0003d12f -[(%esp,1)
deleteCharactersInRange:]
+277 0001af59 4e decl %esi

+278 0001af5a a1a8700300 movl
0x000370a8,%eax length
+283 0001af5f 89442404 movl %eax,
0x04(%esp)
+287 0001af63 893c24 movl %edi,
(%esp)
+290 0001af66 e8c4210200 calll
0x0003d12f -[(%esp,1) length]
+295 0001af6b 46 incl %esi
+296 0001af6c 39c6 cmpl %eax,
%esi
+298 0001af6e 728e jb
0x0001aefe

```

This next part is actually related to the previous part (I just split them up so they wouldn't take as much space in this document).

Here it checks for any other characters like dashes ("-") in our entered license key and removes them. It loops thru the entire length of the serial to check this (thats why it calls [length])

```

+412 0001d124 3c400004 lis r2,0x4
+416 0001d128 3c600004 lis r3,0x4
+420 0001d12c 8083a574 lwz r4,0xa574(r3)
stringWithString:
+424 0001d130 8062adbc lwz r3,0xadbc(r2)
NSString
+428 0001d134 7fa5eb78 or r5,r29,r29
+432 0001d138 4bfeff03 bla 0xffffeff00 +
[NSString stringWithString:]
+436 0001d13c 3c400004 lis r2,0x4
+440 0001d140 8082a0a8 lwz r4,0xa0a8(r2)
length
+444 0001d144 7c7e1b78 or r30,r3,r3
+448 0001d148 4bfeff03 bla 0xffffeff00 -[r3
length]
+452 0001d14c 28030014 cmplwi r3,0x14
+456 0001d150 408201b8 bne 0x1d308
return;

```

```

+300 0001af70 a174750300 movl
0x00037574,%eax stringWithString:
+305 0001af75 8b0dbc7d0300 movl
0x00037dbc,%ecx NSString
+311 0001af7b 89442404 movl %eax,
0x04(%esp)
+315 0001af7f 890c24 movl %ecx,
(%esp)
+318 0001af82 897c2408 movl %edi,
0x08(%esp)
+322 0001af86 e8a4210200 calll
0x0003d12f +[NSString stringWithString:]
+327 0001af8b 89c6 movl %eax,
%esi
+329 0001af8d a1a8700300 movl
0x000370a8,%eax length
+334 0001af92 89442404 movl %eax,
0x04(%esp)
+338 0001af96 893424 movl %esi,
(%esp)
+341 0001af99 e891210200 calll
0x0003d12f -[(%esp,1) length]
+346 0001af9e 83f814 cmpl
$0x14,%eax
+349 0001afa1 740a je
0x0001afad

```

Here we put our "cleaned" serial in a new string and compare the length of it to 14 (thats a hexadecimal value which converted to a decimal value is 20)

```

+460 0001d154 3fa00004 lis r29,0x4
+464 0001d158 809da570 lwz r4,0xa570(r29)
substringToIndex:
+468 0001d15c 3b800008 li r28,0x8
+472 0001d160 7fc3f378 or r3,r30,r30
+476 0001d164 7f85e378 or r5,r28,r28
+480 0001d168 4bfeff03 bla 0xffffeff00 -[r3
substringToIndex:]
+484 0001d16c 7c7b1b78 or r27,r3,r3
+488 0001d170 3c400004 lis r2,0x4
+492 0001d174 8082a56c lwz r4,0xa56c(r2)
substringFromIndex:
+496 0001d178 3f400004 lis r26,0x4
+500 0001d17c 7fc3f378 or r3,r30,r30
+504 0001d180 7f85e378 or r5,r28,r28
+508 0001d184 4bfeff03 bla 0xffffeff00 -[r3
substringFromIndex:]
+512 0001d188 809aaa50 lwz r4,0xaa50(r26)
lowercaseString
+516 0001d18c 3fc00004 lis r30,0x4
+520 0001d190 4bfeff03 bla 0xffffeff00 -[r3
lowercaseString]
+524 0001d194 7c7c1b78 or r28,r3,r3
+528 0001d198 3c400004 lis r2,0x4
+532 0001d19c 8082a6fc lwz r4,0xa6fc(r2)
stringWithCapacity:
+536 0001d1a0 807eae44 lwz r3,0xae44(r30)
NSMutableString
+540 0001d1a4 38a00064 li r5,0x64
+544 0001d1a8 4bfeff03 bla 0xffffeff00 +
[NSMutableString stringWithCapacity:]
+548 0001d1ac 7c7e1b78 or r30,r3,r3
+552 0001d1b0 3f400004 lis r26,0x4
+556 0001d1b4 3c400004 lis r2,0x4
+560 0001d1b8 809aa6f4 lwz r4,0xa6f4(r26)
appendString:
+564 0001d1bc 38a28fe0 addi r5,r2,0x8fe0 BX
+568 0001d1c0 4bfeff03 bla 0xffffeff00 -[r3
appendString:]
...OUTPUT REMOVED...
+812 0001d2b4 4bfeff03 bla 0xffffeff00 -[r3
stringByAppendingString:]
+816 0001d2b8 4bfffba5 bl _CLDigestFromString
+820 0001d2bc 809da570 lwz r4,0xa570(r29)
substringToIndex:
+824 0001d2c0 83daa0bc lwz r30,0xa0bc(r26)
isEqualToString:
+828 0001d2c4 38a0000c li r5,0xc
+832 0001d2c8 4bfeff03 bla 0xffffeff00 -[r3
substringToIndex:]
+836 0001d2cc 7c651b78 or r5,r3,r3
+840 0001d2d0 7f83e378 or r3,r28,r28
+844 0001d2d4 7fc4f378 or r4,r30,r30
+848 0001d2d8 4bfeff03 bla 0xffffeff00 -[r3
isEqualToString:]

```

```

+361 0001afad a170750300 movl 0x00037570,%eax
substringToIndex:
+366 0001afb2 89442404 movl %eax,0x04(%esp)
+370 0001afb6 c744240808000000 movl
$0x00000008,0x08(%esp)
+378 0001afbe 893424 movl %esi,(%esp)
+381 0001afc1 e869210200 calll 0x0003d12f
-[(%esp,1) substringToIndex:]
+386 0001afc6 89c7 movl %eax,%edi
+388 0001afc8 a16c750300 movl 0x0003756c,%eax
substringFromIndex:
+393 0001afcd 89442404 movl %eax,0x04(%esp)
+397 0001afd1 c744240808000000 movl
$0x00000008,0x08(%esp)
+405 0001afd9 893424 movl %esi,(%esp)
+408 0001afdc e84e210200 calll 0x0003d12f
-[(%esp,1) substringFromIndex:]
+413 0001afe1 8b0d507a0300 movl 0x00037a50,%ecx
lowercaseString
+419 0001afe7 894c2404 movl %ecx,0x04(%esp)
+423 0001afeb 890424 movl %eax,(%esp)
+426 0001afee e83c210200 calll 0x0003d12f
-[(%esp,1) lowercaseString]
+431 0001aff3 89c6 movl %eax,%esi
+433 0001aff5 a1fc760300 movl 0x000376fc,%eax
stringWithCapacity:
+438 0001affa 8b0d447e0300 movl 0x00037e44,%ecx
NSMutableString
+444 0001b000 89442404 movl %eax,0x04(%esp)
+448 0001b004 890c24 movl %ecx,(%esp)
+451 0001b007 c744240864000000 movl
$0x00000064,0x08(%esp) 'd'
+459 0001b00f e81b210200 calll 0x0003d12f
+[NSMutableString stringWithCapacity:]
+464 0001b014 89c3 movl %eax,%ebx
+466 0001b016 a1f4760300 movl 0x000376f4,%eax
appendString:
+471 0001b01b 89442404 movl %eax,0x04(%esp)
+475 0001b01f c7442408b8670300 movl
$0x000367b8,0x08(%esp) BX
+483 0001b027 891c24 movl %ebx,(%esp)
+486 0001b02a e800210200 calll 0x0003d12f
-[(%esp,1) appendString:]
...OUTPUT REMOVED...
+761 0001b13d e8ed1f0200 calll 0x0003d12f
-[(%esp,1) appendString:]
+766 0001b142 a1f8700300 movl 0x000370f8,%eax
stringByAppendingString:
+771 0001b147 89442404 movl %eax,0x04(%esp)
+775 0001b14b 895c2408 movl %ebx,0x08(%esp)
+779 0001b14f 893c24 movl %edi,(%esp)
+782 0001b152 e8d81f0200 calll 0x0003d12f
-[(%esp,1) stringByAppendingString:]
+787 0001b157 890424 movl %eax,(%esp)
+790 0001b15a e891fbffff calll _CLDigestFromString
+795 0001b15f 8b0d70750300 movl 0x00037570,%ecx
substringToIndex:
+801 0001b165 894c2404 movl %ecx,0x04(%esp)
+805 0001b169 c74424080c000000 movl
$0x0000000c,
0x08(%esp)
+813 0001b171 890424 movl %eax,(%esp)
+816 0001b174 e8b61f0200 calll 0x0003d12f
-[(%esp,1) substringToIndex:]
+821 0001b179 8b0dbc700300 movl 0x000370bc,%ecx
isEqualToString:
+827 0001b17f 894c2404 movl %ecx,0x04(%esp)
+831 0001b183 89442408 movl %eax,0x08(%esp)
+835 0001b187 893424 movl %esi,(%esp)
+838 0001b18a e8a01f0200 calll 0x0003d12f
-[(%esp,1) isEqualToString:]

```

Here it takes the first 8 characters of our string, makes it lowercase and adds “BXRE&{3? GD>XhyBsDUUK=” (all the [appendString:] calls that I removed from the output does this). This is then sent to something called _CLDigestFromString. By quickly examining this function in our otx output it appears it is simply used to make an md5 hash out of the string sent to it.

The first 12 characters from the hash is then compared to the last 12 characters of the entered license key ([substringToIndex:] and [isEqualToString:] calls). If they match we have a valid license key!

Reverse

So here is the basic scheme for the app:

1. A 8 character long seed with only uppercase hexadecimal characters is needed.

Example: 1234ABCD

2. We add BXRE&{3?GD>XhyBsDUUK= to the end of our seed.

Example: 1234ABCDBXRE&{3?GD>XhyBsDUUK=

3. We make an md5 hash out of it

Example: 1234ABCDBXRE&{3?GD>XhyBsDUUK= -> 7cf80d726bd936751a46704f40b88b27

4. We take the first 12 characters of our hash and add them to our seed

Example: 1234ABCD7CF80D726BD9

5. We have a valid serial! :)

There you have it folks! We have reversed a simple and weak registration scheme by just examining our output and what different Objective-C calls were made. It's not always this easy but I made this tutorial as a starting point before we get our hands dirty and explore dirty little things like reversing actual assembler code...(in a later tutorial, I promise! ;))

Homework

1. Practice/Learn your C/Objective-C/Java/Perl/AppleScript or whatever language you prefer -skillz and write the actual keygen! (Easy/Hard)