

Forum:[Hack Forums](#)

Thread: [\[Mac\]\[Tutorial\] Setting up and cracking!](#)

Krillere - January 4, 2012, 1:10 pm

Intro

Hello, and welcome to my tutorial. If you – like me – are a Mac user, who's been looking to get into Reverse Engineering, you've probably already Google'd around, finding nothing but useless stuff (Although, a few Mac RE's actually exists. I'll link my favourite in the 'Credits'). Well, I sat down one day, and just began reading, and suddenly, I had a working setup and a CrackMe solved.

So, here's my tutorial, GL & HF with it. :-)

Disclaimer

What you do with this knowledge, I am NOT responsible for. The example included is a simple CrackMe, but should You choose to crack some expensive software, I am not to blame if something happens to you! I simply shared some knowledge.

The method I'm using, is just one of many. This is just to get you started. You should be able to find another way to crack it when you're done here.

Also, this is my first post in this section, and I'm still feeling unsure about alot of this stuff, so feel free to correct me.

Software

Well, do solve Part 2, we're going to need some software. First of all, you should have XCode installed. It's a tool for Apple Developers, but AFAIK it's available for everyone else too.

Next, you're going to need OTX. There's multiple version available loads of places on the internet, but you'll need the latest. IF you download one of the older versions, you wont have access to disassembling x64 binaries. You can download the latest from here: Quote:

<http://otx.osxninja.com/builds/trunk/>

using SVN.

You'll need to compile this yourself in XCode, but that's as easy as opening the .xcodeproject file, and 'Archive' the application, and share it to your destination of choice.

Now you'll need Class-Dump, this can be downloaded from here: Quote:

<http://www.codethecode.com/projects/class-dump/>

We're also going to use GDBInit. A nifty script created by one of the most helpfull Mac RE's I've seen. This tool can be downloaded from here: Quote: <http://reverse.put.as/gdbinit/>

Be sure to download the newest version. When this is done, you'll have to move it into your user's folder, and rename it .gdbinit. This can be done via Terminal like this: Quote:

```
mv ~/Downloads/gdbinitVERSION ~/.gdbinit
```

Of course, you replace the 'gdbinitVERSION'-part with the filename of the file you downloaded. And the path of it's not in the Download-directory.

Another great tool to get is 'Hopper Disassembler'. It's on the Apple Mac AppStore, and is worth every penny (Actually, it's pretty cheap!). If you can't afford this, GDB will do just fine for now.

You could choose to use IDA instead, but that's really your choice. I chose to use Hopper and GDB for doing this. :-)

Using GDB

For this CrackMe we'll be using GDB debugger. This comes with XCode (Or OS X, not sure TBH). This is used in the Terminal.

There's a few things you should know about GDB. When using gdbinit, we'll have to first start GDB, like this:

Quote:gdb

Now we'll choose our 'target' file, with the following command:

Quote:exec-file ~/MyFolder/MyFolder2/File.app/Contents/MacOS/File

As you see, the .app file you normally launch, is actually just a folder. I'm not going to explain how this works, but just that you'll find the 'real' executable in the Contents/MacOS/ folder.

To find out what's actually happening while debugging the app, we'll have to set some breakpoints. This can be done like this:

Quote:break 0x11

But, you can use functions, lines and other methods too. So if you want to know when the app is checking the length of a string, you can use the following: Quote:break [NSString length]

Really, if you know how Objective-C works, this should be easy enough. :-)

To continue when the breakpoint has been reached, just write 'c' or 'continue', and the app will continue execution.

We're also going to have to actually run the app. This can be done by either writing 'r' or 'run'. When the app is being paused by a breakpoint, you can exit the app by writing 'kill', or simply CMD+Q the app while it's running.

We're also able to read from these addresses. With a simple command like this: Quote:x/s *0xADDRESS

This'll output a string.

A list of other types:

Spoiler ([Click to View](#))

- o(octal)
- x(hex)
- d(decimal)
- u(unsigned decimal)
- t(binary)
- f(float)
- a(address)
- i(instruction)
- c(char)
- s(string)
- T(OSType)

But looking at the app isn't enough for us. We want to change some values too. This can be done with the 'set' command. It's actually as simple as it seems. Can be done like this: Quote: set \$eax = (int) 0x12345

Of course, this can be done with addresses too.

We're also going to use the 'cfz' command, which'll make the je, jne and so on not to be followed. Simply skipping the jump.

For most of these methods, theres more options that what I listed, but this is the basics, and it's all we need to solve the CrackMe.

If you want to know more about GDB, look at the help, or go Google around.

The cracking part

Well, seems like we're finally ready to crack something! Yay!

The CrackMe we're going to use, can be downloaded from here: Spoiler ([Click to View](#))
<http://reverse.put.as/wp-content/uploads/2010/05/MSJ20091.zip>

I made my own CrackMe too, if you guys want it, I'll upload it to so you can have a go at it. (Although, it's VERY easy!).

We're going to need to know what we're up against, so what we want to do, is to find all functions and global variables. This can be accomplished by using class-dump. The following command should do it: Quote: class-dump "Path/To/File/Challenge #1.app/Contents/MacOS/Challenge #1" >> classdump.txt

This will dump the variables and functions in a file named classdump.txt. This will be in your current working folder (seen by using 'pwd' in Terminal).

When we open this file, we see a few functions. My classdump.txt looks like this:

Spoiler ([Click to View](#)) Quote: /

Generated by class-dump 3.3.4 (64 bit).

class-dump is Copyright © 1997-1998, 2000-2001, 2004-2011 by Steve Nygard.

*/

#pragma mark -

/

File: /Users/Christian/Desktop/Challenge #1.app/Contents/MacOS/Challenge #1
UUID: B4FB5C47-4F51-F65D-E63B-9B5652A2346D
Arch: Intel 80x86 (i386)

* Objective-C Garbage Collection: Unsupported

*/

@interface Level1 : NSObject

```
{
    id errorSheet;
    id incorrectSerialSheet;
    id mainWindow;
    id nameField;
    id registeredSheet;
    id serialField;
    id welcomeSheet;
    BOOL hasBeenRegistered;
}
```

- (void)awakeFromNib;

- (void)applicationDidFinishLaunching:(id)arg1;
- (void)continueWelcomeButton:(id)arg1;
- (void)quitCorrectSerialButton:(id)arg1;
- (void)okErrorSheetButton:(id)arg1;
- (void)okIncorrectSerialButton:(id)arg1;
- (void)cancelButton:(id)arg1;
- (void)unregisterButton:(id)arg1;
- (void)emailResults:(id)arg1;
- (void)verifyRegistration:(id)arg1;
- (BOOL)isRegistered;
- (BOOL)validateSerial:(id)arg1 forName:(id)arg2;
- (BOOL)applicationShouldTerminateAfterLastWindowClosed:(id)arg1;

@end

The method 'verifyRegistration:' seems like something we could use.

Now we want to get the ASM-code of this application. It's time to use OTX. Open OTX, and drag the binary file into the window. It should recognize it, and you should be able to save the dump.

This file is a little longer, so I'm not going to post the content, but it should be the same.

In most ASM-dumps, you're probably not going to have such 'good' output. Everything I get my hands on, I'll have to find the addresses using another app (Hopper), and then find the address, but in this case, we got the method names before the ASM code, which allows us to search for our function. Now search for Quote: -(void)[Level1 verifyRegistration:]

and you'll get to the function that gets called when you press the 'Register'-button.

I assume you know some ASM before beginning this, so you should be able to spot where the action happens. But if not, here's the 'if'-statement that we're going to tamper with:

Spoiler ([Click to View](#))Quote:+174 0000293c e81d270000 calll 0x0000505e -[(%esp,1) validateSerial:forName:]

```
+179 00002941 84c0 testb %al,%al
+181 00002943 0f84f500 je 0x00002a3e
+187 00002949 a100400 movl 0x00004000,%eax standardUserDefaults
+192 0000294e 89442404 movl %eax,0x04(%esp)
```

Start up Terminal, load up GDB and the Challenge #1 as previously explained..

Now we want to set a breakpoint on the address, so that we can stop it from jumping to the 'bad' message.

This is done by using this: Spoiler ([Click to View](#))Quote:break *0x00002943

Now we'll run the app, simply by calling 'run'.

The app will open, and you'll be presented with a window telling you about the rules and so. Continue, and you'll see 'Name' and 'Serial' textfields. Write whatever you want, and press the 'Register'-button. This will pause the app and should focus the Terminal window.

Write 'cfz', press enter and the crack is actually done. But since our app is still in pause, we have to tell it to continue running, by using the 'continue' or 'c' command.

Now the app will tell us that it's registered successfully, GJ! :-)

Now, as said in the 'Disclaimer', this is not the only way, but it's by far the easiest IMO.

Credits

There's only one guy that I'll show you, it's fG.

This guy made a tutorial, that everyone is able to follow, and the blog is worth following. He's also the author of various tools, including gdbinit. You can find multiple CrackMe's on his site, and links to other sites.

The blog can be found here: <http://reverse.put.as/>

The End

If you found any errors or something weird in the tutorial, just let me know, and I'll fix it ASAP. Feel free to ask questions, and I'll do my best to answer them. :-)

- Krillere

bottomy - January 4, 2012, 2:31 pm

Good tutorial, it's nice to see something related to Mac OS X come up in this section.

And you should definitely post the crackme you made, since even if it's easy at least it'll be something (no OS X challenges have been posted yet, well not as far as I know of). I'm also going to work on some OS X challenges when I get the time.

And by the way what does GDBInit do?

hacku4eva - January 4, 2012, 3:26 pm

cool tut! might open up lion os now and give this a go

jeseravi - January 4, 2012, 8:37 pm

+rep!!

really helpful. I was having trouble with this :D

bottomy - January 4, 2012, 11:05 pm

Just had a look at the crackme now (it's actually more correctly a keygenme). So I whipped up a quick little keygen for it, mind you it's not very good since I'm in the middle of doing something else so I didn't really bother breaking down the algorithms but instead just ripping it's asm or converting it to an equivalent.

Spoiler ([Click to View](#)) If you use an OS prior to Lion, just create a normal NSAutoreleasePool.

```
Code void GenerateSerial(NSString *Username)
{
    @autoreleasepool {
        const char *Name = [Username UTF8String]; //The keygenme is actually using lossyCString, but
        that's deprecated and I'm too lazy to look up the proper equivalent/suggested equivalent.
```

```
        uint32_t Res = 0, Val, ecx = 0;
        size_t Length = strlen(Name);
        for (size_t Loop = 0; Loop < Length; Loop++)
        {
            Val = Name[Loop] * ((Loop + 1) * 4);
            Val = (Val << 4) - Val;
```

```
//Ripped the asm code, too lazy to workout the actual algorithm.
```

```

asm("movl %0,%%ecx\n"
"\tmovl %1,%%edx" : : "g" (ecx), "g" (Val) : "ecx", "edx");

asm("movl $0x68db8bad,%eax\n"
"\tleal 0x0029a(%edx,%ecx),%ecx\n"
"\timull %ecx\n"
"\tmovl %ecx,%eax\n"
"\tsarl $0x1f,%eax\n"
"\tsarl $0x0c,%edx\n"
"\tsubl %eax,%edx\n"
"\tmovl %ecx,%eax\n"
"\timull $0x00002710,%edx,%edx\n"
"\tsubl %edx,%eax");

asm("movl %%ecx,%0\n"
"\tmovl %%eax,%1" : "=g" (ecx), "=g" (Res) : : "ecx", "eax");
}

```

```

NSString *Serial = [NSString stringWithFormat: @"%i", Res];

```

```

ecx = 0;
for (size_t Loop = 0; Loop < Length; Loop++)
{
    Val = Name[Loop] * (4 + (Loop * 8));
    Val += 45 + ((Val + (Val * 4)) * 2);
}

```

//Ripped the asm code, too lazy to workout the actual algorithm.

```

asm("movl %0,%%ecx\n"
"\tmovl %1,%%edx" : : "g" (ecx), "g" (Val) : "ecx", "edx");

asm("movl $0x68db8bad,%eax\n"
"\tadd %edx,%ecx\n"
"\timull %ecx\n"
"\tmovl %ecx,%eax\n"
"\tsarl $0x1f,%eax\n"
"\tsarl $0x0c,%edx\n"
"\tsubl %eax,%edx\n"
"\tmovl %ecx,%eax\n"
"\timull $0x00002710,%edx,%edx\n"
"\tsubl %edx,%eax");

asm("movl %%ecx,%0\n"
"\tmovl %%eax,%1" : "=g" (ecx), "=g" (Res) : : "ecx", "eax");
}

```

```

Serial = [Serial stringByAppendingFormat: @"%i", Res];
printf("Serial: %s\n", [Serial UTF8String]);
}
}

```

Example input "GenerateSerial(@"bottomy");" and would print "Serial: 59426451"

Alternatively it wouldn't be very hard to patch the keygenme to display the serial in the serial text box. Would just add the path code after the two loops, and would need to remove the very first exit

check (checks if the serials length is equal to 8 or not). If someones interested in seeing an example of that, I could do it a bit later.

Krillere - January 5, 2012, 9:17 am

Bottomy, that would be great, any OSX-action is good. GDBInit adds some functionality to GDB, but it seems to have screwed with some stuff in XCode(Not 100% sure, though), so be carefull!

Also, I could do another tutorial on how to patch this with a Hex-editor. But if you're going to do it, I'm not going to bother :P

Just one more thing. I love the way you did the keygen thing. Haven't tried it yet, but I'm sure as hell going to try something like that. Gj! :-)

Black_Mamba - January 5, 2012, 5:54 pm

Good job! Can you make something like this for the personal computer windows?

T-rom - January 5, 2012, 6:01 pm

Thank you for this :)

I am really hoping mac people will start jumping in here more :)

Krillere - January 5, 2012, 9:01 pm

@ Black_Mamba: Well, it should be more or less the same, but with OllyDBG, IDA or something. I don't really use Windows, so I'm not sure about how to do things on Windows. :-)

@ T-rom: Thanks, that's what I'm hoping for too. :-)

bottomy - January 6, 2012, 6:29 pm

(Jan 5, 2012 10:17 AM)Krillere Wrote: [CITE](#) Also, I could do another tutorial on how to patch this with a Hex-editor. But if you're going to do it, I'm not going to bother :P

Well I was going to patch it so it would display the serial (so in a sense it would then be the keygen), not patch it to crack it. Anyway I just gave it a shot, but I'm rather stumped on something.

I changed the jump after the serial length check to nops, so you don't need to enter a serial with length of 8 to get the serial. And then I added my main patch code from when it starts splitting the serial you enter to where the checks end. So I append the two parts, and then get self's NSTextField instance variable for serial (well it actually turned out to be for the name, since I'm hardcoding the lookup at fixed offsets instead of doing a dynamic lookup using the runtime. So didn't actually notice I had the wrong instance till recently lol), and then sending it the message "setStringValue:" with the appended serial as the value. However it says "setStringValue:" isn't a correct selector for NSTextField (along with raising an exception), which it clearly is, so yeh rather stumped.

Code (gdb)

```
0x00002bed in -[Level1 validateSerial:forName:] ()
```

```
1: x/i $pc 0x2bed <-[Level1 validateSerial:forName:]+335>:  mov    %eax,(%esp)
```

```
(gdb) po $eax
```

```
<NSTextField: 0x665e00>
```

```
(gdb) continue
```

Continuing.

```
2012-01-07 12:52:23.368 Challenge #1[13566:1307] -[NSTextField setStringValue:]: unrecognized selector sent to instance 0x665e00
```

```
2012-01-07 12:52:23.370 Challenge #1[13566:1307] -[NSTextField setStringValue:]: unrecognized selector sent to instance 0x665e00
```

```
2012-01-07 12:52:23.371 Challenge #1[13566:1307] (
```

```

0 CoreFoundation 0x9804fd87 __raiseError + 231
1 libobjc.A.dylib 0x9b95c149 objc_exception_throw + 155
2 CoreFoundation 0x98053390 -[NSObject doesNotRecognizeSelector:] + 256
3 CoreFoundation 0x97fa2059 ____forwarding____ + 457
4 CoreFoundation 0x97fa1e22 _CF_forwarding_prep_0 + 50
5 Challenge #1 0x00002c05 -[Level1 validateSerial:forName:] + 359
6 Challenge #1 0x00002941 -[Level1 verifyRegistration:] + 179
7 CoreFoundation 0x97fa5091 -[NSObject performSelector:withObject:] + 65
8 AppKit 0x9af1dcb3 -[NSApplication sendAction:to:from:] + 232
9 AppKit 0x9af1db90 -[NSControl sendAction:to:] + 102
10 AppKit 0x9af1da93 -[NSCell _sendActionFrom:] + 160
11 AppKit 0x9af1ce50 -[NSCell trackMouse:inRect:ofView:untilMouseUp:] + 2295
12 AppKit 0x9afa0351 -[NSButtonCell trackMouse:inRect:ofView:untilMouseUp:] + 501
13 AppKit 0x9af1b893 -[NSControl mouseDown:] + 943
14 AppKit 0x9aee47b1 -[NSWindow sendEvent:] + 7533
15 AppKit 0x9ae7c6ff -[NSApplication sendEvent:] + 4788
16 AppKit 0x9ae0dc82 -[NSApplication run] + 1007
17 AppKit 0x9b0a218a NSApplicationMain + 1054
18 Challenge #1 0x0000240e _start + 216
19 Challenge #1 0x00002335 start + 41

```

)

Yet if I call `[$eax stringValue: @"blah"]` through GDB's command line, it works. Call it while it's paused at `0x00002bed`, so `eax` holds the `NSTextField` object.

Anyway, I'll have a look at it again later, I probably made some stupid mistake.

Krillere - January 7, 2012, 5:07 am

Seems like you're a really great at this, Bottomy. Would love to see some more stuff from you. Also, if you have some spare-time some day, would you mind commenting some of your keygen code? Seems like something I would love to learn. :-)

bottomy - January 7, 2012, 12:56 pm

(Jan 7, 2012 06:07 AM)Krillere Wrote: [CITE](#) Seems like you're a really great at this, Bottomy. Would love to see some more stuff from you. Also, if you have some spare-time some day, would you mind commenting some of your keygen code? Seems like something I would love to learn. :-)

I'm not really sure what I can say about it, since it was mainly just a rip (only a little bit was changed). So it's basically just the operations from `-[Level1 validateSerial:forName:]` that we care about. While some of it is converted to Obj-C, the rest (the main algorithmic part) is just an assembly rip. Since I couldn't be bothered working out what it was in a higher level/original context, it's most like a divide or possibly a modulo.

It'll make more sense if you look at `-[Level1 validateSerial:forName:]`. I'll post the otool dump of it, with some comments I added to make it a bit easier to understand.

Code `-[Level1 validateSerial:forName:]`:

```

00002a9e pushl  %ebp
00002a9f movl  %esp,%ebp
00002aa1 pushl  %edi
00002aa2 pushl  %esi
00002aa3 pushl  %ebx
00002aa4 subl  $0x3c,%esp
00002aa7 movl  0x00004040,%eax ;length
00002aac movl  %eax,0x04(%esp)

```

```

00002ab0  movl  0x10(%ebp),%eax ;Serial
00002ab3  movl  %eax,(%esp)
00002ab6  calll 0x0000505e ; symbol stub for: _objc_msgSend
00002abb  cmpl  $0x08,%eax ;compare [Serial length] with 8
00002abe  jne  0x00002c4e
00002ac4  movl  0x00004044,%eax ;lossyCString
00002ac9  movl  $0x004,%esi
00002ace  xorl  %ebx,%ebx
00002ad0  movl  %eax,0x04(%esp)
00002ad4  movl  0x14(%ebp),%eax ;Name
00002ad7  movl  %eax,(%esp)
00002ada  calll 0x0000505e ; symbol stub for: _objc_msgSend
00002adf  movl  %eax,(%esp)
00002ae2  movl  %eax,%edi
00002ae4  calll 0x00005063 ; symbol stub for: _strlen
00002ae9  xorl  %ecx,%ecx
00002aeb  movl  %eax,0xd0(%ebp)
00002aee  jmp  0x00002b27 ;Entering first loop, which creates the first half of the serial
00002af0  movsbl (%edi,%ebx),%eax ;eax = c_Name[ebx++] (by c_Name I'm referring to the
return of performing lossyCString on the Name object)
00002af4  incl  %ebx
00002af5  imull %esi,%eax ;eax *= 4 (plus 4 for each iteration)
00002af8  addl  $0x04,%esi ;esi += 4
00002afb  movl  %eax,%edx
00002afd  shll  $0x04,%edx ;edx <= 4
00002b00  subl  %eax,%edx ;edx -= eax
00002b02  movl  $0x68db8bad,%eax
00002b07  leal  0x0029a(%edx,%ecx),%ecx
00002b0e  imull %ecx
00002b10  movl  %ecx,%eax
00002b12  sarl  $0x1f,%eax
00002b15  sarl  $0x0c,%edx
00002b18  subl  %eax,%edx
00002b1a  movl  %ecx,%eax
00002b1c  imull $0x00002710,%edx,%edx
00002b22  subl  %edx,%eax
00002b24  movl  %eax,0xe0(%ebp)
00002b27  cmpl  %ebx,0xd0(%ebp)
00002b2a  ja  0x00002af0

00002b2c  movl  0xe0(%ebp),%eax
00002b2f  movl  $0x004,%esi
00002b34  xorl  %ebx,%ebx
00002b36  movl  $0x00003078,0x08(%esp) ;@"%i"
00002b3e  movl  %eax,0x0c(%esp) ;result of the previous algorithm
00002b42  movl  0x00004028,%eax ;stringWithFormat:
00002b47  movl  %eax,0x04(%esp)
00002b4b  movl  0x00004054,%eax ;NSString
00002b50  movl  %eax,(%esp)
00002b53  calll 0x0000505e ; symbol stub for: _objc_msgSend
00002b58  movl  %edi,(%esp) ;name
00002b5b  movl  %eax,0xd8(%ebp)
00002b5e  calll 0x00005063 ; symbol stub for: _strlen

```

```

00002b63 xorl  %ecx,%ecx
00002b65 movl  %eax,0xd4(%ebp)
00002b68 jmp   0x00002b9c ;Entering second loop, which creates the last half of the serial
00002b6a movsbl (%edi,%ebx),%eax
00002b6e incl  %ebx
00002b6f imull %esi,%eax
00002b72 addl  $0x08,%esi
00002b75 leal  (%eax,%eax,4),%edx
00002b78 leal  0x2d(%eax,%edx,2),%edx
00002b7c movl  $0x68db8bad,%eax
00002b81 addl  %edx,%ecx
00002b83 imull %ecx
00002b85 movl  %ecx,%eax
00002b87 sarl  $0x1f,%eax
00002b8a sarl  $0x0c,%edx
00002b8d subl  %eax,%edx
00002b8f movl  %ecx,%eax
00002b91 imull $0x00002710,%edx,%edx
00002b97 subl  %edx,%eax
00002b99 movl  %eax,0xe4(%ebp)
00002b9c cmpl  0xd4(%ebp),%ebx
00002b9f jb   0x00002b6a

```

```

00002ba1 movl  0xe4(%ebp),%eax
00002ba4 movl  $0x004,%esi
00002ba9 xorl  %ebx,%ebx
00002bab movl  $0x00003078,0x08(%esp) ;@"%i"
00002bb3 movl  $0x004,%edi
00002bb8 movl  %eax,0x0c(%esp) ;result of the previous algorithm
00002bbc movl  0x00004028,%eax ;stringWithFormat:
00002bc1 movl  %eax,0x04(%esp)
00002bc5 movl  0x00004054,%eax ;NSString
00002bca movl  %eax,(%esp)
00002bcd calll 0x0000505e ; symbol stub for: _objc_msgSend
00002bd2 movl  %ebx,0x08(%esp) ;(NSRange){ 0, 4 } (this includes next instruction)
00002bd6 movl  %esi,0x0c(%esp)
00002bda movl  %eax,0xdc(%ebp)
00002bdd movl  0x00004048,%eax ;substringWithRange:
00002be2 movl  %eax,0x04(%esp)
00002be6 movl  0x10(%ebp),%eax
00002be9 movl  %eax,(%esp) ;Serial
00002bec calll 0x0000505e ; symbol stub for: _objc_msgSend
00002bf1 movl  %esi,0x08(%esp) ;(NSRange){ 4, 4 } (this includes next instruction)
00002bf5 movl  %edi,0x0c(%esp)
00002bf9 movl  %eax,%ebx
00002bfb movl  0x00004048,%eax ; substringWithRange:
00002c00 movl  %eax,0x04(%esp)
00002c04 movl  0x10(%ebp),%eax
00002c07 movl  %eax,(%esp) ;Serial
00002c0a calll 0x0000505e ; symbol stub for: _objc_msgSend

```

```

00002c0f movl  %ebx,(%esp) ;First half of Serial
00002c12 movl  %eax,%esi

```

```

00002c14  movl    0xd8(%ebp),%eax ;First string created from the result of the first algorithm
00002c17  movl    %eax,0x08(%esp)
00002c1b  movl    0x00004008,%eax ; isEqual:
00002c20  movl    %eax,0x04(%esp)
00002c24  calll   0x0000505e ; symbol stub for: _objc_msgSend
00002c29  testb   %al,%al
00002c2b  je      0x00002c4e
00002c2d  movl    0xdc(%ebp),%eax ;Second string created from the result of the second
algorithm
00002c30  movl    %esi,(%esp) ;Second half of Serial
00002c33  movl    %eax,0x08(%esp)
00002c37  movl    0x00004008,%eax ; isEqual:
00002c3c  movl    %eax,0x04(%esp)
00002c40  calll   0x0000505e ; symbol stub for: _objc_msgSend
00002c45  movl    $0x001,%edx
00002c4a  testb   %al,%al
00002c4c  jne     0x00002c50
00002c4e  xorl    %edx,%edx
00002c50  addl    $0x3c,%esp
00002c53  movl    %edx,%eax
00002c55  popl    %ebx
00002c56  popl    %esi
00002c57  popl    %edi
00002c58  leave
00002c59  ret

```

Krillere - January 7, 2012, 1:54 pm

I'll have a look at it, thanks :-)

I was wondering if you could point me towards a book or something equal to teach me some assembler? I seem to be having a hard time understanding some of it :-/

bottomy - January 7, 2012, 2:13 pm

I haven't read any, so wouldn't know what to suggest. Since I was self taught, only thing I did use was an instruction reference. This is the reference I use <http://ref.x86asm.net/> though you should also find a site that goes into more detail on individual instructions, this is more so for other extensions (such as SSE) since a number of its instruction's operations act differently depending on the input (since it makes it more flexible).

It's also good to read the Intel and AMD manuals, and especially the System V ABI manual from AMD for 64bits (since those are the conventions that OS X now follows).

Krillere - January 7, 2012, 2:54 pm

I'll look into it. Asked a few of my teachers for some books too, so hopefully that'll be enough. :-)

BlackHat D3c0d3r - December 19, 2013, 8:09 pm

Gud Job Bro..

Nice Tutz.. ?

..It HELPS A Lot..

?