

Reverse engineering and patching of iOS app. How to protect our app from hacker attacks.

In this post i will share with you

- **how to decrypt an app store binary**
- **basic steps to reverse engineer an iOS app**
- **how to alter application code, add new code, create a patch for the given iOS app, with the help of the flexible iOS runtime**

Why someone would need this information ?

As iOS developers, we are often faced with the task about making our apps more secure. By having the mentioned above knowledge, one can **be aware of the majority of ways a hacker can use to attack his app, thus he can create more secure apps**. Each individual step depends on those prior to it, so they have to be executed in the given order. **This post would be a very common path for ios hackers and security researchers.**

Agenda:

1. **Setting up the enviroment**
 - a. **for decrypting**
 - b. **for altering code**
2. **Decrypt mach-o binary from app store app.**
3. **Gather information about class and methods names from encrypted mach-o binary.**
 - a. **tool used: [class-dump](#)**
4. **Alter implementation of some instance method.**
 - a. **tool used: [theos](#)**

Lets start by setting up our enviroment.

Firstly, we would need a jailbroken device. My previos [post](#) explains what is "jailbreak" and how to achieve it on device running iOS 10. In addition to that, this time we have to enable our device for ssh sessions. The way to do this in iOS 10 is really well described in this [short video](#). Furthermore, you can install **SFTPEabler** from **Cydia**, in order to transfer files to your device over ssh.

NOTE: *the default ssh password for the root user of iOS is "alpine". One can change it by issuing "passwd" command during his ssh session.*

Secondly, we need mac computer with installed XCode, connected on the same network with the iOS device.

Next we have to copy over a patched version of lldb's debugserver from XCode to our device.

debugserver executable can be found inside the disk image located at
"/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/
DeviceSupport/7.0.3<choose_the_latest_ios_version>/DeveloperDiskImage.dmg".

So mount that image, and inside it, debugserver executable can be found in "/usr/bin". Copy it to some other place, for example in your home directory. With the help of debugserver, we can attach to each process that is running on our device and debug it, but before that we have to code sign it with special entitlements, which will allow him to attach to processes. Here are the entitlements:

entitlements.plist

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
<key>com.apple.springboard.debugapplications</key>
<true/>
<key>run-unsigned-code</key>
<true/>
<key>get-task-allow</key>
<true/>
<key>task_for_pid-allow</key>
<true/>
</dict>
</plist>
```

If we saved them inside a document named "**entitlements.plist**" again in our home directory, then we can codesign the executable that we copied inside our home directory by issuing the following command in terminal:

code signing debugserver

```
codesign -s - --entitlements ~/entitlements.plist -f ~/debugserver
```

Once we have that done, we can then copy this newly signed version of debugserver to our device's file system. We want that file inside **/usr/bin** directory in our iOS device. Later we will use it to decrypt mach-o executable files from app store apps. With that we are ready with the setup for decrypting! Lets prepare our device for altering code.

On the device, open Cydia and install the following tools:

- LD64 (a.k.a ldid) ("Link Identity Editor") - tool for modifying a binary's entitlements easily. ldid also generates SHA1 hashes for the binary signature, so the device kernel executes the binary.
- Perl
- Git
- Class-Dump
- Make
- LLVM+Clang
- Darwin CC Tools
- Core Utilities
- Core Utilities (/bin)

Warning: latest version of Perl package places the perl executable in "/usr/local/bin/", but for our setup we will need this executable to be in usr/bin. So simply copy the file with the command : "cp /usr/local/bin/perl /usr/bin"

All of the above tools are needed in order to run one special tool for developing stuff for iOS from the device itself. That tool is called "Theos". Theos was initially "iphone-framework", a project created to simplify building code at the command line for iOS devices (primarily jailbroken devices). It later underwent significant changes and became Theos, a flexible Make-based build system primarily for jailbreak software development.

So choose a path at which you will install theos on your device. In this blog post i will install theos inside ~/theos directory.

1. make ~ the active directory: "cd ~"
2. clone the git repo inside ~/theos :

```
git clone --recursive https://github.com/theos/theos.git
```

- a. If either of cURL or wget complains about some SSL certificate stuff, you can temporarily disable SSL verification in git by issuing

```
git config --global http.sslVerify false
```

- i. Don't forget to enable SSL verification after that because you will be otherwise exposed to man in the middle attacks

3. Create THEOS enviroment variable, which will be used when compiling and loading code alternations:

```
export THEOS=var/root/theos // /var/root/ is the root's home dir ~
```

4. Download one sdk from <https://sdks.website> , extract it and place it inside ~/theos/sdks
 - a. for example - theos/sdks/iPhoneOS9.3.sdk/

In order to test Theos installation, change the active dir to some place other than \$THEOS/bin. I will use ~/code_alternation.

So

```
mkdir ~/code_alternation  
cd ~/code_alternation
```

run the following bash script:

```
$THEOS/bin/nic.pl
```

and you should see the following screen

```
stoyan.stoyanov — ssh root@192.168.2.2 — 47x17
[Stoans-iPhone:~/tweaks root# $THEOS/bin/nic.pl ]
NIC 2.0 - New Instance Creator
-----
[ 1.] iphone/activator_event
[ 2.] iphone/application_modern
[ 3.] iphone/cydyget
[ 4.] iphone/Flipswitch_switch
[ 5.] iphone/Framework
[ 6.] iphone/ios7_notification_center_widget
[ 7.] iphone/library
[ 8.] iphone/notification_center_widget
[ 9.] iphone/preference_bundle_modern
[10.] iphone/tool
[11.] iphone/tweak
[12.] iphone/xpc_service
Choose a Template (required): █
```

if not then you haven't set up your THEOS enviroment variable to the right location. Create a project of your choice (i will go with iphone/tweak), cd in to the project's directory and issue command

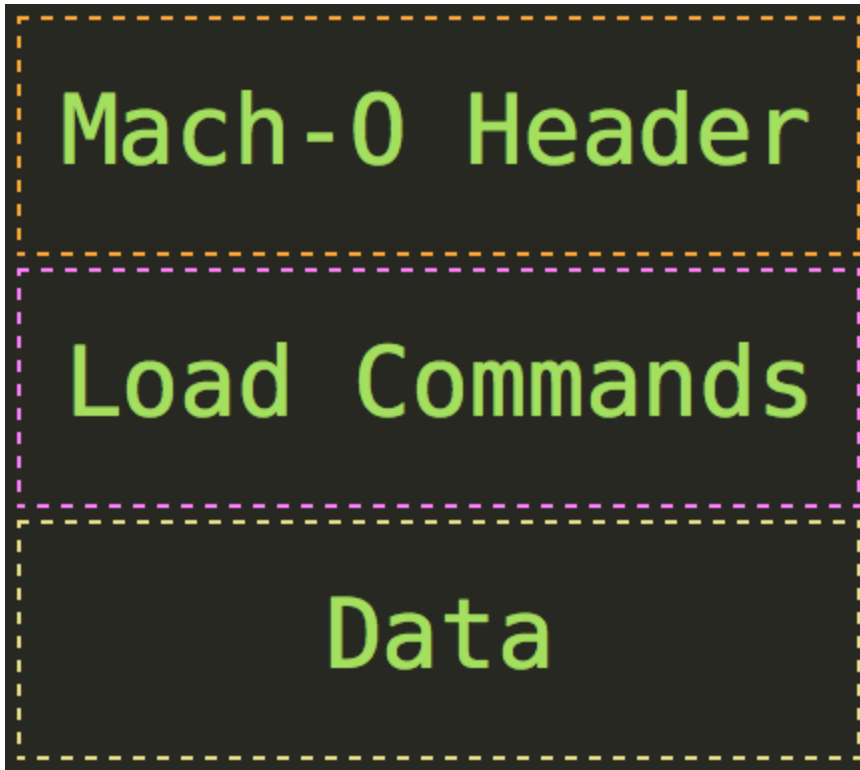
```
make update-theos
```

If everything finishes fine with no errors, you have setup your enviroment correctly! You can start decrypting apps and create patches for them.

Now lets have a look at how to decrypt iOS Mach-O binary.

App store binaries are decrypted with the [Apple's FairPlay DRM](#). Generally speaking Mach-o binaries, can be divided into two groups - for multiple architectures and for single architecture. In context of iOS, both of those types are used. The CPU architectures of iOS devices are armv6, armv7, armv7s, arm64, so if you want to build an app for the newest and older iphones simultaneously, then the app will be compiled for different architectures, thus creating universal executable file. Those multi-architecture binaries are also known as "*fat binary*" because they are containing your code compiled several times for different architectures ("*fattened*"). Wait, but how a hacker knows of what type is the mach-o file of my app ? There is one macOS command line tool called "otool", that comes with the install of XCode. With the help of that tool anyone can inspect what kind of binary he is dealing with.

We can think of universal binaries like if they are collection of multiple single architecture binaries. Each single architecture binary has three main parts, as we can see from the image below.



The first part, contains meta information, encryption information, various offsets. Second part contains definitions of all functions inside our program, and the third part contains the implementations of those functions, and other resources.

Now when we know this information we can proceed with decompilation of app.

I will decompile [Vbox7.com](#) app. The process will be identical for all other apps.

Decompilation steps:

- download the app on the prepared device.
- copy the executable file of the app on your mac desktop. In Vbox7 case, the executable is called VBOX7_iOS and is located in

```
/var/containers/Bundle/Application/7D3867BD-0DA4-4847-9266-3829C6617431/  
VBOX7_iOS.app/VBOX7_iOS
```

One tip to find this path is to run the app, and in your ssh session to type "ps -ax" which will list all of the running processes, and the path to their executables

```
stoyan.stoyanov — ssh root@192.168.2.2 — 180x46
30311 ?? 0:24.34 /Applications/Music.app/Music
30352 ?? 6:25.63 //Applications/IFile.app/IFile_
30355 ?? 0:02.77 /usr/libexec/gamecontrollerd
30495 ?? 2:55.02 /System/Library/PrivateFrameworks/AggregateDictionary.framework/Support/aggregate
30597 ?? 0:00.24 /usr/libexec/wcd
30666 ?? 0:00.81 /private/var/containers/Bundle/Application/E1698768-E9CA-492D-8482-32BCD16D80B6/YouTube.app/PlugIns/NotificationServiceExtension.appex/NotificationService
30916 ?? 0:00.09 /System/Library/Frameworks/LocalAuthentication.framework/Support/coreauthd
30970 ?? 0:03.16 /System/Library/PrivateFrameworks/SoftwareUpdateServices.framework/Support/softwareupdateservicecd
31313 ?? 0:00.12 /System/Library/Frameworks/Metal.framework/XPCServices/MTLCompilerService.xpc/MTLCompilerService
31366 ?? 0:00.03 /System/Library/CoreServices/ReportCrash.com.apple.ReportCrash.Jetsam
31397 ?? 0:00.07 /usr/libexec/UTATaskingAgent server-init
31408 ?? 0:00.02 /usr/libexec/nlsagent
31412 ?? 0:00.29 /System/Library/CoreServices/EscrowSecurityAlert.app/EscrowSecurityAlert
31417 ?? 0:03.59 /System/Library/CoreServices/CacheDeleteDaily
31421 ?? 0:00.02 /usr/local/bin/dropbear -F -R -p 22
31423 ?? 0:00.01 /usr/libexec/rocketd
31442 ?? 0:00.18 /usr/libexec/afcd
31445 ?? 0:00.07 /usr/libexec/mobile_assertion_agent
31448 ?? 0:02.48 /usr/libexec/installd
31449 ?? 0:00.00 (MSUnrestrictProc)
31454 ?? 0:00.03 /System/Library/PrivateFrameworks/MobileInstallation.framework/XPCServices/com.apple.MobileInstallationHelperService.xpc/com.apple.MobileInstallationHelpe
31457 ?? 0:00.36 /usr/libexec/mobile_installation_proxy
31469 ?? 0:00.08 /System/Library/PrivateFrameworks/CloudServices.framework/XPCServices/com.apple.sbd.xpc/com.apple.sbd
31475 ?? 0:05.71 /usr/libexec/SafariCloudHistoryPushAgent
31480 ?? 0:00.19 /usr/libexec/pipelined
31487 ?? 0:00.24 /System/Library/PrivateFrameworks/MapsSupport.framework/mapspushd
31493 ?? 0:00.24 /Applications/MobileGest.app/MobileGest
31495 ?? 0:01.02 /System/Library/PrivateFrameworks/NotificationCenter.framework/NotificationCenter
31531 ?? 0:04.09 /var/containers/Bundle/Application/87891773-CSFA-4D6D-939B-390979F260DE/Skype.app/Skype
31533 ?? 0:09.94 /Applications/MobileMail.app/MobileMail
31568 ?? 0:00.36 /Applications/Contacts.app/PlugIns/ContactsCoreSpotlightExtension.appex/ContactsCoreSpotlightExtension
31572 ?? 0:00.28 /System/Library/PrivateFrameworks/UpNextWidget.framework/PlugIns/UpNext.appex/UpNext
31574 ?? 0:00.46 /Applications/Weather.app/PlugIns/WeatherAppTodayWidget.appex/WeatherAppTodayWidget
31590 ?? 0:07.03 /usr/libexec/coreduetd
31621 ?? 0:00.19 /usr/local/bin/dropbear -F -R -p 22
31622 ?? 0:00.01 /usr/libexec/sftp-server
31641 ?? 0:00.28 /usr/local/bin/dropbear -F -R -p 22
32246 ?? 0:00.14 /System/Library/PrivateFrameworks/AssetCacheServices.framework/XPCServices/AssetCacheLocatorService.xpc/AssetCacheLocatorService -d
32249 ?? 0:00.07 /usr/libexec/online-auth-agent
32251 ?? 0:00.12 /usr/libexec/suicd
32258 ?? 0:00.20 /System/Library/PrivateFrameworks/CloudDocsDaemon.framework/XPCServices/ContainerMetadataExtractor.xpc/ContainerMetadataExtractor
32264 ?? 0:00.04 /usr/libexec/networkserviceproxy
32341 ?? 0:03.26 /var/containers/Bundle/Application/7038678D-0D44-4B47-9266-3829C6617431/VBOX7_iOS.app/VBOX7_iOS
31645 ttye000 0:00.06 sh
32344 ttye000 0:00.00 ps -ex
Stoyan-iPhone:- root#
```

- examine VBOX7_iOS mach-o with otool on your mac

```
airPro — -bash — 76x26
[sStoyanov-iMac:airPro stoyan.stoyanov$ otool -fh VBOX7_iOS
Fat headers
fat_magic 0xcafebabe
nfat_arch 2
architecture 0
    cputype 12
    cpusubtype 9
    capabilities 0x0
    offset 16384
    size 5780848
    align 2^14 (16384)
architecture 1
    cputype 16777228
    cpusubtype 0
    capabilities 0x0
    offset 5799936
    size 6838480
    align 2^14 (16384)
Mach header
    magic cputype cpusubtype caps filetype ncmds sizeofcmds flags
    0xfeedface 12 9 0x00 2 75 7296 0x00218085
Mach header
    magic cputype cpusubtype caps filetype ncmds sizeofcmds flags
    0xfeedfacf 16777228 0 0x00 2 75 7960 0x00218085
5
sStoyanov-iMac:airPro stoyan.stoyanov$
```

- we will decrypt the first architecture. You have to choose which architecture to decrypt based what is the architecture your on your device. **Remember the offset for this subbinary.**
- observe the encryption data of subbinary which we choose to decrypt.

```
airPro — -bash — 70x8
[sStoyanov-iMac:airPro stoyan.stoyanov$ otool -l VBOX7_iOS | grep crypt]
    cryptoff 16384
    cryptsize 4734976
    cryptid 1
    cryptoff 16384
    cryptsize 5160960
    cryptid 1
sStoyanov-iMac:airPro stoyan.stoyanov$
```

- **remember cryptoff and cryptsize values.**
- open ssh session to your device

- Run vbox7 app
- Find out what is the processId (PID) for vbox7 app with "ps -ax"
- attach the debugger to the PID of vbox7 app

```
debugserver *:1010 -a <pid_of_vbox7> // every other port than 1010
can be used
```

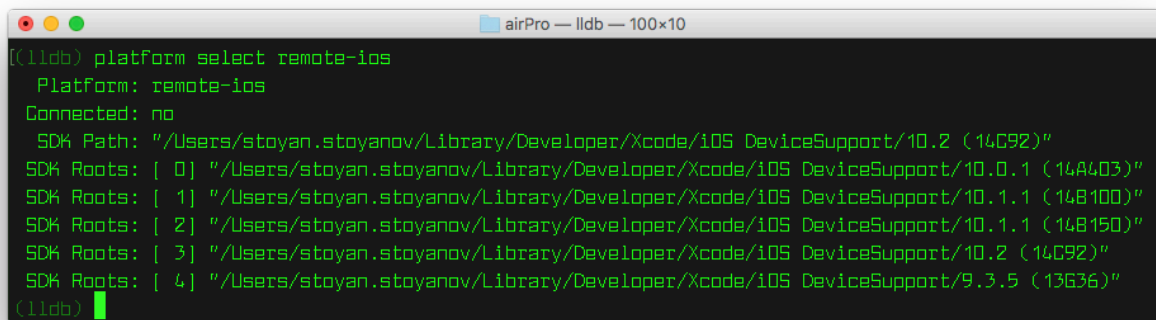
```
32341 ??          3:58.32 /var/containers/Bundle/Application/7D3B67BD-0DA4-4847-9266-3B29C6617431/VBOX7_iOS.app/VBOX7_iOS
31645 ttys000    0:00.11 -sh
32347 ttys000    0:00.00 ps -ax
Stoans-iPhone:~ root# debugserver *:1010 -a 32341
debugserver-@(#)PROGRAM:debugserver  PROJECT:debugserver-360.0.26.1
For arm64.
Attaching to process 32341...
Listening to port 1010 for a connection from *...
```

- on your mac, open terminal and type

```
lldb
```

- to start lldb in interactive mode
- select appropriate platform

```
platform select remote-ios
```



```
airPro - lldb - 100x10
(lldb) platform select remote-ios
Platform: remote-ios
Connected: no
SDK Path: "/Users/stoyan.stoyanov/Library/Developer/Xcode/iOS DeviceSupport/10.2 (14C92)"
SDK Roots: [ 0] "/Users/stoyan.stoyanov/Library/Developer/Xcode/iOS DeviceSupport/10.0.1 (14A403)"
SDK Roots: [ 1] "/Users/stoyan.stoyanov/Library/Developer/Xcode/iOS DeviceSupport/10.1.1 (14B100)"
SDK Roots: [ 2] "/Users/stoyan.stoyanov/Library/Developer/Xcode/iOS DeviceSupport/10.1.1 (14B150)"
SDK Roots: [ 3] "/Users/stoyan.stoyanov/Library/Developer/Xcode/iOS DeviceSupport/10.2 (14C92)"
SDK Roots: [ 4] "/Users/stoyan.stoyanov/Library/Developer/Xcode/iOS DeviceSupport/9.3.5 (13G36)"
(lldb)
```

- connect lldb to your device

```
platform connect connect://192.168.2.2:1010 // here replace the ip
adress and port to yours
```



```
airPro — lldb — 99x13
(lldb) platform connect connect://192.168.2.2:1010
Platform: remote-ios
Triple: arm64-apple-ios
OS Version: 10.2.0
Hostname: (null)
Connected: yes
SDK Path: "/Users/stoyan.stoyanov/Library/Developer/Xcode/iOS DeviceSupport/10.2 (14C92)"
SDK Roots: [ 0] "/Users/stoyan.stoyanov/Library/Developer/Xcode/iOS DeviceSupport/10.0.1 (14A403)"
SDK Roots: [ 1] "/Users/stoyan.stoyanov/Library/Developer/Xcode/iOS DeviceSupport/10.1.1 (14B100)"
SDK Roots: [ 2] "/Users/stoyan.stoyanov/Library/Developer/Xcode/iOS DeviceSupport/10.1.1 (14B150)"
SDK Roots: [ 3] "/Users/stoyan.stoyanov/Library/Developer/Xcode/iOS DeviceSupport/10.2 (14C92)"
SDK Roots: [ 4] "/Users/stoyan.stoyanov/Library/Developer/Xcode/iOS DeviceSupport/9.3.5 (13G36)"
(lldb)
```

- set debug target so that we can get the loaded images list

```
target create VBOX7_iOS
```

```
(lldb) target create VBOX7_iOS
Current executable set to 'VBOX7_iOS' (arm64).
(lldb)
```

- find the start address in memory of the loaded subbinary and remember it

```
image list VBOX7_iOS
```

```
(lldb) image list VBOX7_iOS
[ 0] 45326CC6-C5B1-32A4-88B4-71BAF6AED539 0x00000000100000000
/Users/stoyan.stoyanov/Desktop/airPro/VBOX7_iOS
(lldb)
```

- dump the memory to file from start location *cryptoff+start_address*, with length *cryptsize*

```
memory read --force --outfile ./decrypted.bin --binary --count
5160960 0x00000000100000000+16384
```

```
airPro — lldb — 103x5
(lldb) memory read --force --outfile ./decrypted.bin --binary --count 5160960 0x00000000100000000+16384
5160960 bytes written to './decrypted.bin'
(lldb)
```

Now we can disconnect lldb.

The "decrypted.bin" file is containing decrypted subbinary for the architecture that we have chosen to decrypt. The only thing that left to be done is to override the encrypted bytes of original binary with the one we dumped from the memory and set change cryptId flag to 0 of our subbinary header, which will indicate that there is no encryption.

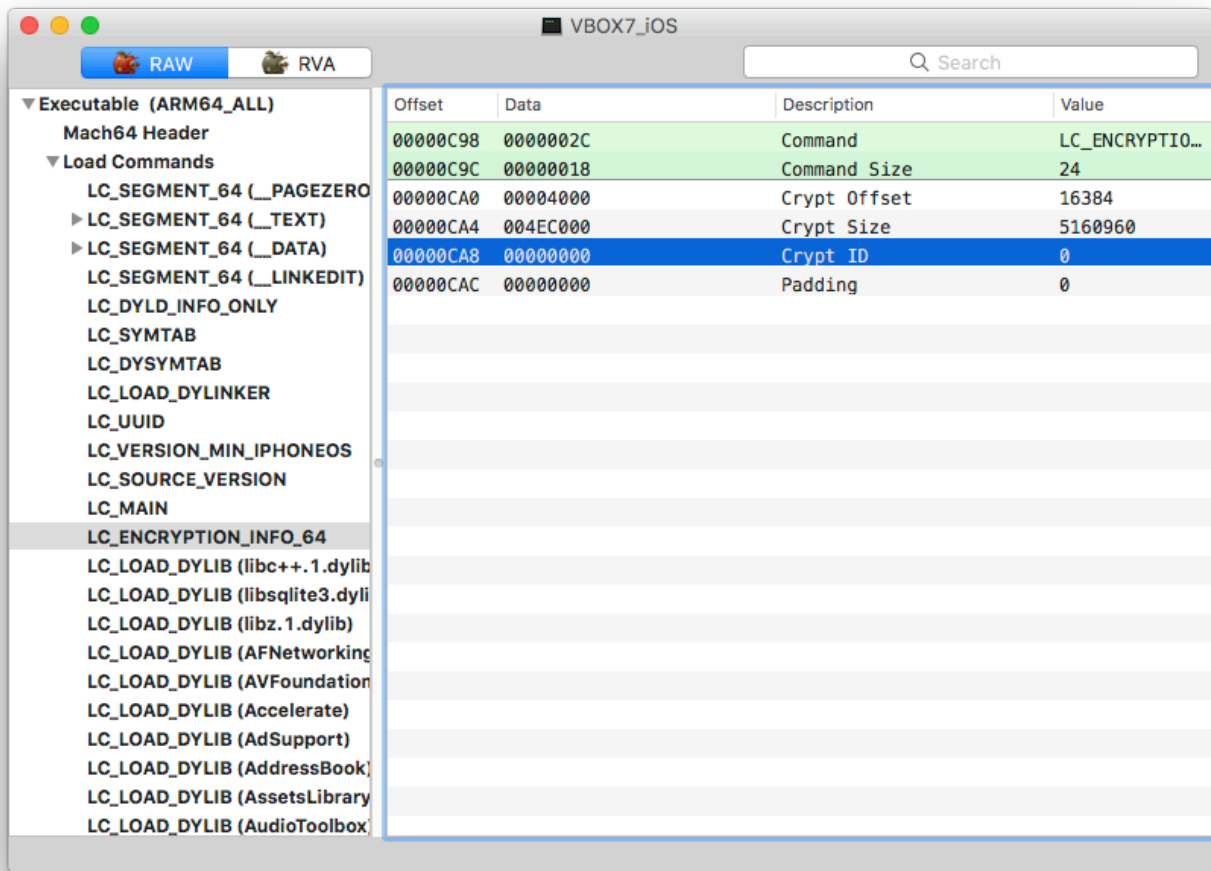
On your mac in your active directory with VBox7_iOS and decrypted.bin type

```
dd seek=16384 bs=1 conv=notrunc if=./decrypted.bin of=./VBox7_iOS //
where 16384 is the start address of subbinary for the architecture we
are decrypting.
```

```
airPro — -bash — 106x5
[sStoyanov-iMac:airPro stoyan.stoyanov$ dd seek=16384 bs=1 conv=notrunc if=./decrypted.bin of=./VBox7_iOS ]
5160960+0 records in
5160960+0 records out
5160960 bytes transferred in 6.306731 secs (818326 bytes/sec)
sStoyanov-iMac:airPro stoyan.stoyanov$
```

The last thing we need to change is one flag inside `_LC_ENCRYPTION_INFO` called `cryptId`, which marks encryption status of the

binary. cryptId value should be 1 initially which will indicate that it the binary is encrypted, and now we need to change it to 0, so that it can be properly handled after we have encrypted him. The easiest way to change this value is through [MachOView](#):



And with that we have successfully decrypted this binary for the specific architecture!

We can now extract information about classes and functions from it. For this purpose we will use a tool called [class-dump](#). Basically, it is a command-line utility for examining runtime information stored in Mach-O files. It generates declarations for the classes, categories and protocols.

For our purpose we use it with options -H and -o

```
./class-dump -H -o ./dumpedHeaders VBOX7_iOS
```

```
Desktop -- -bash -- 117x5
[sStoyanov-iMac:Desktop stoyan.stoyanov$ ./class-dump -H -o ./dumpedHeaders VBoxX7_iOS
2017-03-21 17:05:02.685 class-dump[3616:4B3280] Warning: Parsing instance variable type failed, _backgroundUpdateTask
2017-03-21 17:05:02.693 class-dump[3616:4B3280] Warning: Parsing instance variable type failed, _backgroundTaskId
sStoyanov-iMac:Desktop stoyan.stoyanov$
```

now the directory `"/.dumpedHeaders"` should contain all of the headers of vbox7 app. With this information we can draw a simple scheme that describes the application logic.

The last thing that we are going to do is to alter some of the code of the vbox7 app.

We have all the needed information to achieve this task.

The method that we are going to use for code modifying relies on one jailbreak tweak creation platform called "*Cydia Substrate*". It is formerly known as "*Mobile Substrate*" and it provides a way to develop iOS add-ons. **Basically we are going to create a "tweak" in the form of dynamic library .dlib** for our app which will provide new implementation to some instance methods inside vbox7 classes.

Lets start!

- ssh into your device
- choose a directory in which you will develop your tweak.
- go inside that dir, and run

```
$THEOS/bin/nic.pl
```

- select a template for iphone/tweak
- type in a project name
- type in a bundle identifier(package name)
- type the bundle identifier of the app your dynamic library should attach(very important)
- type which processes you want to be killed after successful install of your tweak

```
stoyan.stoyanov -- ssh root@192.168.2.2 -- 128x24
[Stoans-iPhone:~/tweaks root# $THEOS/bin/nic.pl
NIC 2.0 - New Instance Creator
-----
[1.] iphone/activator_event
[2.] iphone/application_modern
[3.] iphone/cydyget
[4.] iphone/flipswitch_switch
[5.] iphone/framework
[6.] iphone/ios7_notification_center_widget
[7.] iphone/library
[8.] iphone/notification_center_widget
[9.] iphone/preference_bundle_modern
[10.] iphone/tool
[11.] iphone/tweak
[12.] iphone/xpc_service
[Choose a Template (required): 11
[Project Name (required): vbox7Patch
[Package Name [com.yourcompany.vbox7patch]: com.stoyan.vbox7patch CAuthor/Maintainer Name [System Administrator]: Stoyan Stoyanov]
[iphone/tweak] MobileSubstrate Bundle Filter [com.apple.springboard]: com.vbox7
[iphone/tweak] List of applications to terminate upon installation (space-separated, '-' for none) [SpringBoard]:VBoxX7_iOS
Instantiating iphone/tweak in vbox7patch/...
Done.
Stoans-iPhone:~/tweaks root#
```

Now we should write the code to the tweak itself.

Open Tweak.xm file inside your newly created theos project folder with text editor of your choice and program it as you like (you can find reference online). In my case Tweak.xm looks like this:

```
%hook UIViewController
- (void)viewDidAppear:(BOOL)animated {
%orig; // Call through to the original function with its original
arguments.
if ([NSStringFromClass(self.class)
isEqualToString:@"VBBGMusicViewController"]) {

UILabel *label = [[UILabel alloc] initWithFrame:self.view.frame];
[label setBackgroundColor:[UIColor blueColor]];
label.textAlignment = NSTextAlignmentCenter;
label.text = @"example of\nmethod hooking\nby Stoyan";
label.numberOfLines = 0;
[label setFont:[UIFont systemFontOfSize:40]];
label.textColor = [UIColor whiteColor];

self.view = label;
[[self view] layoutIfNeeded];

UIAlertController *alert = [UIAlertController
alertWithTitle:@"example of method hooking by Stoyan"
message:nil preferredStyle:UIAlertControllerStyleAlert];
[alert addAction:[UIAlertAction actionWithTitle:@"Great!"
style:UIAlertActionStyleCancel handler:nil]];
[self presentViewController:alert animated:YES completion:nil];
}
NSLog(@"NSLog - %@", NSStringFromClass(self.class));
}
%end
```

Furthermore i have added this line to the project's makefile (in your case it may not be necessary)

```
Vbox7Enchancer_FRAMEWORKS = UIKit
```

At this point you should be able to build your tweak. Go inside tweak's folder and type

```
make package
```

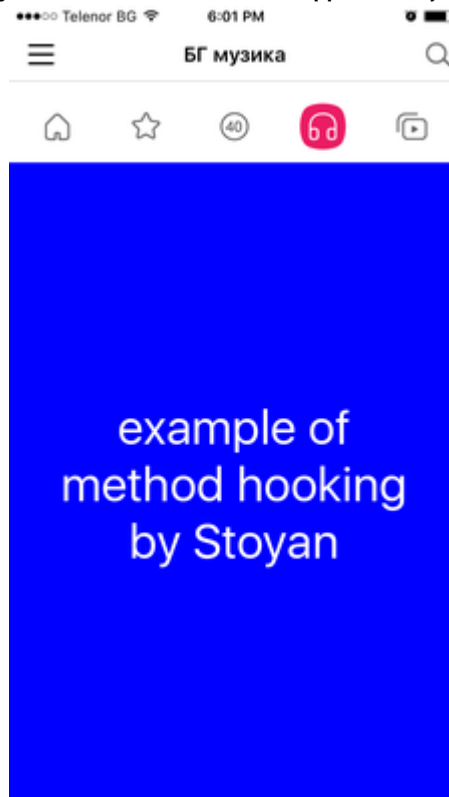
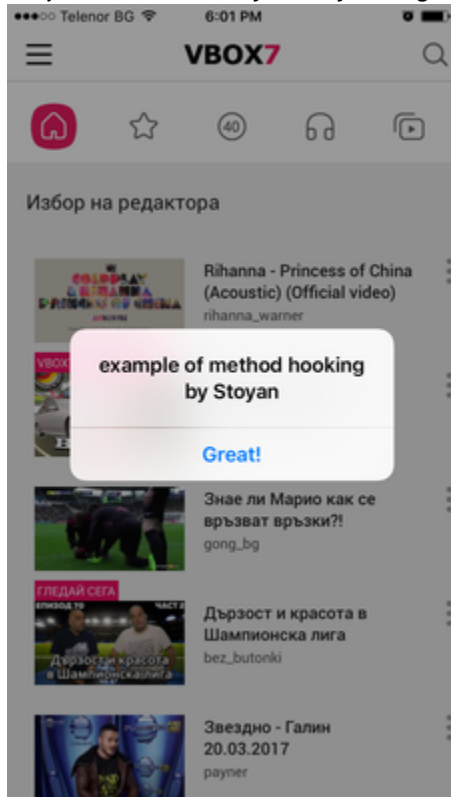
and you should see the project compiling

```
stoyan.stoyanov — ssh root@192.168.2.2 — 127x13
Stoans-iPhone:~/tweaks/vbox7patch root# make package
> Making all for tweak vbox7Patch_
==> Preprocessing Tweak.xm_
==> Compiling Tweak.xm (armv7)_
==> Linking tweak vbox7Patch (armv7)_
==> Preprocessing Tweak.xm_
==> Compiling Tweak.xm (arm64)_
==> Linking tweak vbox7Patch (arm64)_
==> Merging tweak vbox7Patch_
==> Signing vbox7Patch_
> Making stage for tweak vbox7Patch_
dpkg-deb: building package 'com.stoyan.vbox7pa7patch' in './packages/com.stoyan.vbox7Pa7patch_0.0.1-1+debug_iphoneos-arm.deb'.
Stoans-iPhone:~/tweaks/vbox7patch root#
```

Once compiled successfully, you can install it by typing

```
make install
```

and you are done! **Now if you run your target app, your modifications should be applied.** In my case:



So that is how an app can be reverse engineered and patched.

In this sample i have patched one of the UIViewController methods, but in the same way we can modify code dealing with user credentials and private data! **For Swift apps things are a bit different, but the concept is the same, because they are using the same objective c runtime.**

For hybrid apps altering code and reverse engineering are way easier tasks because there is no encryption on the "hybrid source" - .js, html etc. One can just browse in to the app directory, and modify the code inside some js file for example, and the changes will be applied immediately. So be aware of the vulnerabilities and write safe code!