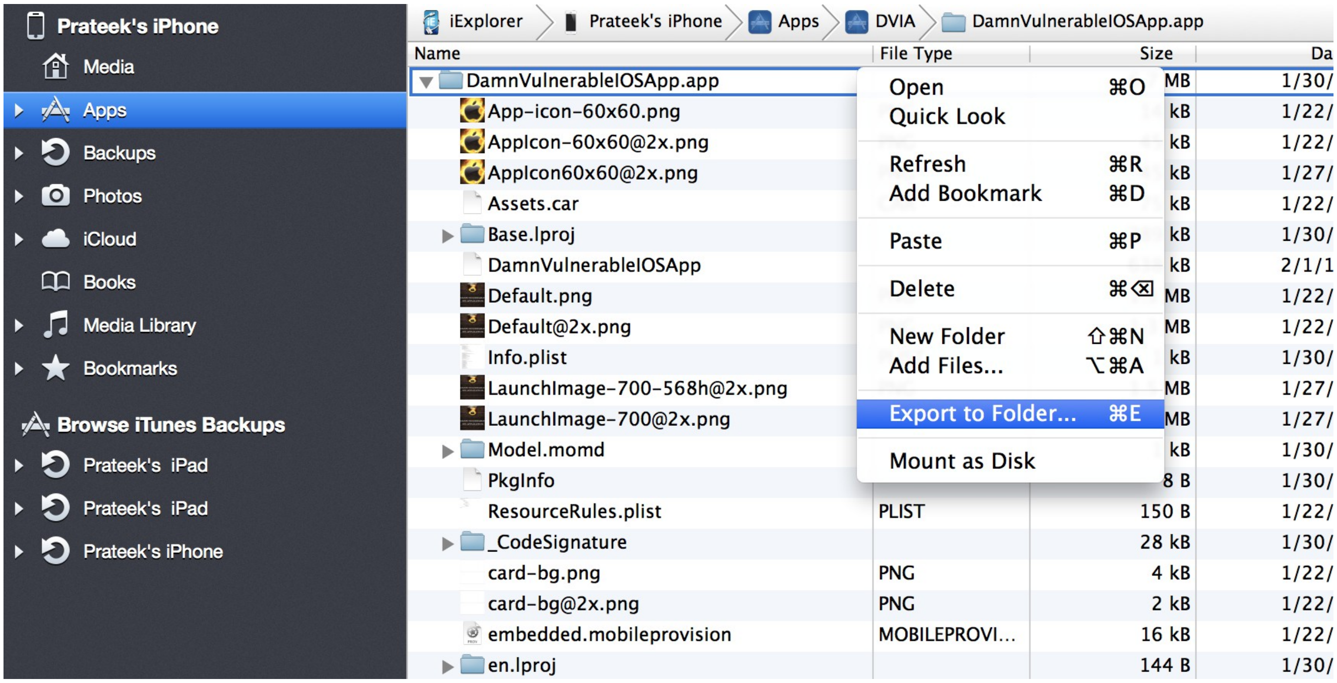
Damn Vulnerable IOS Application Solutions
http://damnvulnerableiosapp.com/

Application Patching – Jailbreak Evasion

For this challenge, we will be fetching the application folder from the device, patching the binary and installing the modified application back to the device. This is because we cannot test the check for a jailbroken device on the simulator.

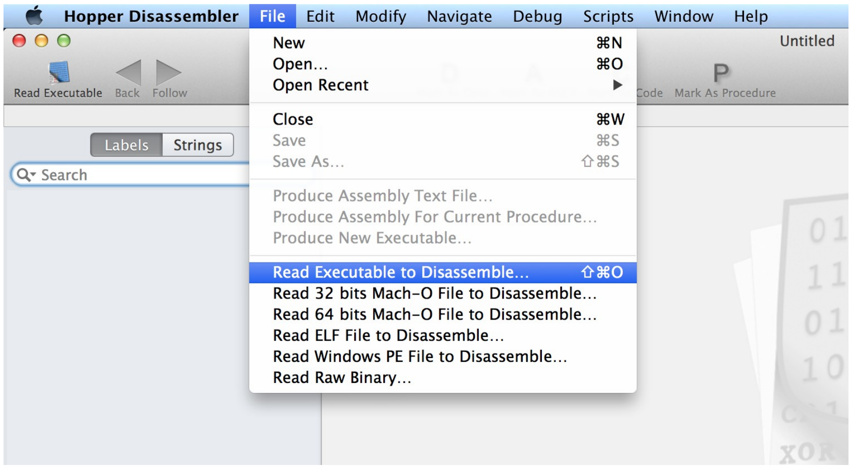
To copy the application folder for DVIA to your system, open iExplorer (make sure your device is connected to your laptop), head over to your device, then the Apps section, select the DVIA App folder, and right click on it to export it to your system.



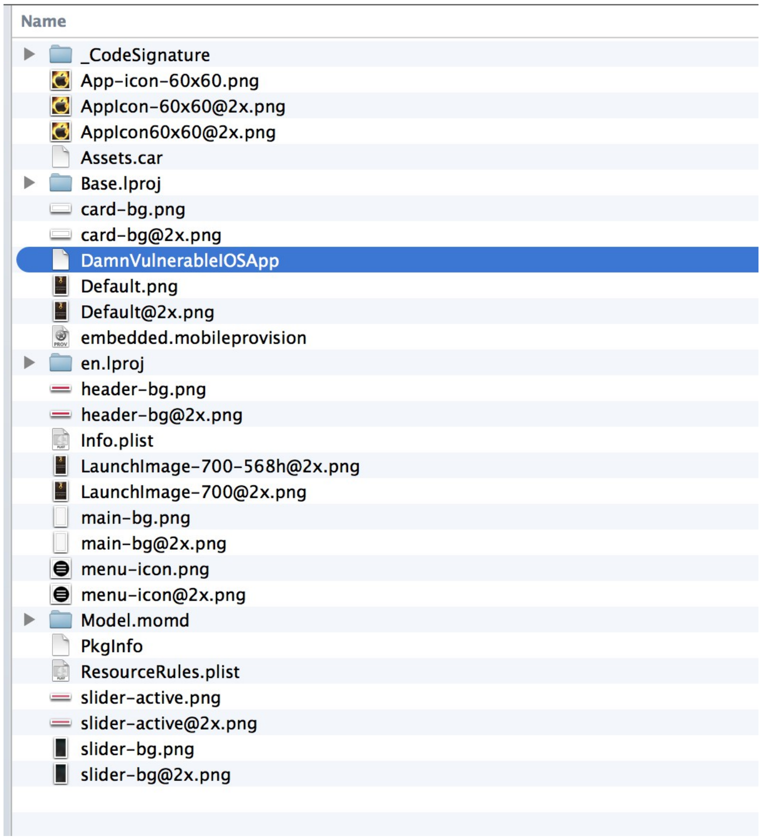
Name	File Type	Size	Da
DamnVulnerableIOSApp.app		MB	1/30/
App-icon-60x60.png		kB	1/22/
AppIcon-60x60@2x.png		kB	1/22/
AppIcon60x60@2x.png		kB	1/27/
Assets.car		kB	1/22/
Base.lproj		kB	1/30/
DamnVulnerableIOSApp		kB	2/1/1
Default.png		MB	1/22/
Default@2x.png		MB	1/22/
Info.plist		kB	1/30/
LaunchImage-700-568h@2x.png		MB	1/27/
LaunchImage-700@2x.png		MB	1/27/
Model.momd		kB	1/30/
PkgInfo		8 B	1/30/
ResourceRules.plist	PLIST	150 B	1/22/
_CodeSignature		28 kB	1/30/
card-bg.png	PNG	4 kB	1/22/
card-bg@2x.png	PNG	2 kB	1/22/
embedded.mobileprovision	MOBILEPROVI...	16 kB	1/22/
en.lproj		144 B	1/30/

The file exported will be named DamnVulnerableIOSApp.app

Now open Hopper and select the option *File → Read Executable To Disassemble.*

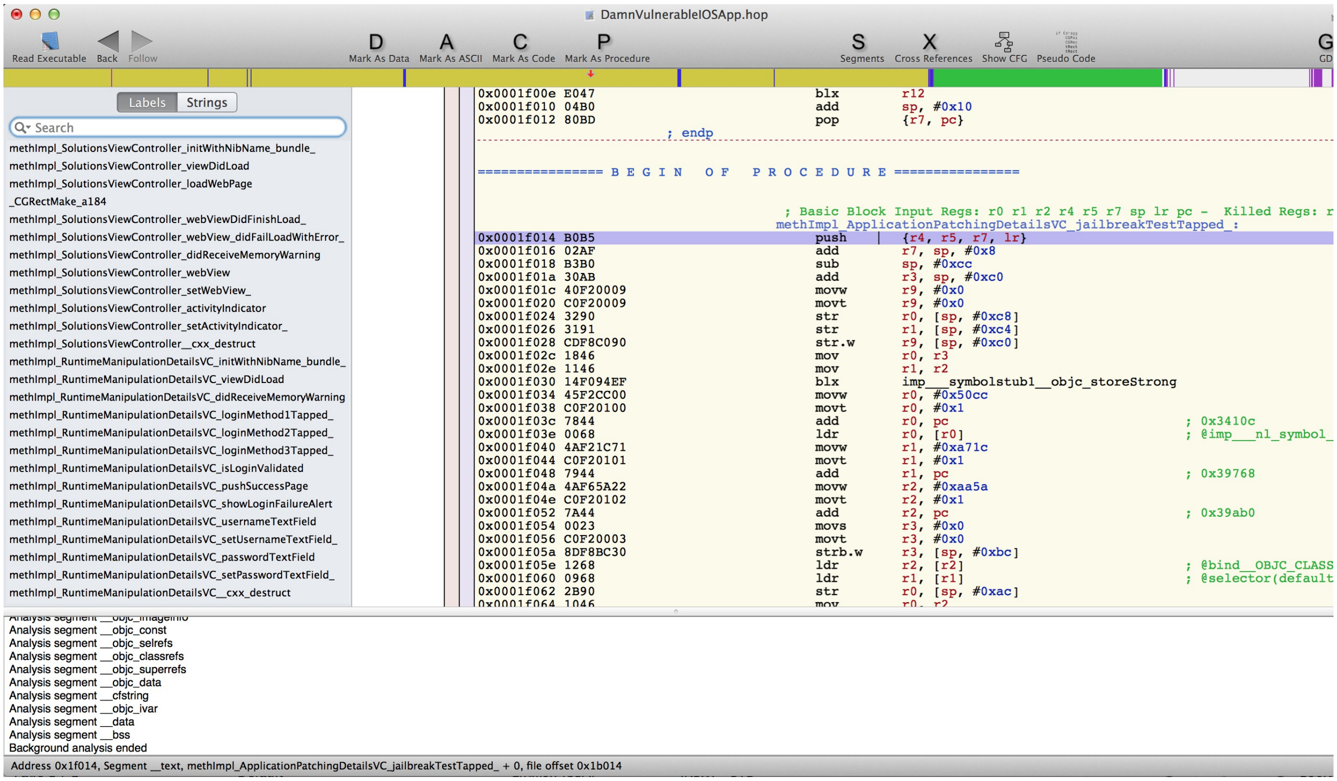

Hopper Disassembler
File
Edit
Modify
Navigate
Debug
Scripts
Window
Help
Untitled
Read Executable
Back
Follow
Code
Mark As Procedure
Labels
Strings
Search
New ⌘N
Open… ⌘O
Open Recent ▶
Close ⌘W
Save ⌘S
Save As… ⇧⌘S
Produce Assembly Text File…
Produce Assembly For Current Procedure…
Produce New Executable…
Read Executable to Disassemble… ⇧⌘O
Read 32 bits Mach-O File to Disassemble…
Read 64 bits Mach-O File to Disassemble…
Read ELF File to Disassemble…
Read Windows PE File to Disassemble…
Read Raw Binary…

Give the Binary from the application folder that we just exported. The application binary will be inside the .app folder.


Name

- _CodeSignature
- App-icon-60x60.png
- AppIcon-60x60@2x.png
- AppIcon60x60@2x.png
- Assets.car
- Base.lproj
- card-bg.png
- card-bg@2x.png
- DamnVulnerableIOSApp
- Default.png
- Default@2x.png
- embedded.mobileprovision
- en.lproj
- header-bg.png
- header-bg@2x.png
- Info.plist
- LaunchImage-700-568h@2x.png
- LaunchImage-700@2x.png
- main-bg.png
- main-bg@2x.png
- menu-icon.png
- menu-icon@2x.png
- Model.momd
- PkgInfo
- ResourceRules.plist
- slider-active.png
- slider-active@2x.png
- slider-bg.png
- slider-bg@2x.png

Hopper will start disassembling the binary and produce an output like this...



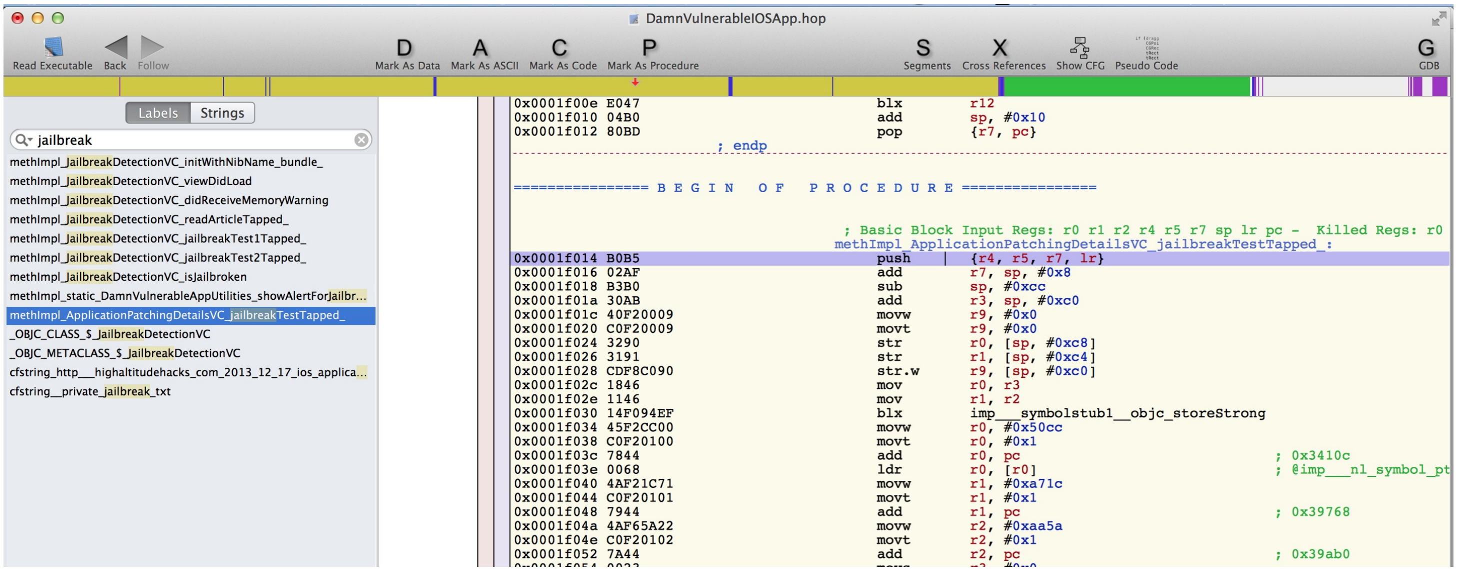
methImpl_SolutionsViewController_initWithNibName_bundle_
methImpl_SolutionsViewController_viewDidLoad
methImpl_SolutionsViewController_loadWebPage
_CGRectMake_a184
methImpl_SolutionsViewController_webViewDidFinishLoad_
methImpl_SolutionsViewController_webView_didFailLoadWithError_
methImpl_SolutionsViewController_didReceiveMemoryWarning
methImpl_SolutionsViewController_webView
methImpl_SolutionsViewController_setWebView_
methImpl_SolutionsViewController_activityIndicator
methImpl_SolutionsViewController_setActivityIndicator_
methImpl_SolutionsViewController__cxx_destruct
methImpl_RuntimeManipulationDetailsVC_initWithNibName_bundle_
methImpl_RuntimeManipulationDetailsVC_viewDidLoad
methImpl_RuntimeManipulationDetailsVC_didReceiveMemoryWarning
methImpl_RuntimeManipulationDetailsVC_loginMethod1Tapped_
methImpl_RuntimeManipulationDetailsVC_loginMethod2Tapped_
methImpl_RuntimeManipulationDetailsVC_loginMethod3Tapped_
methImpl_RuntimeManipulationDetailsVC_isLoginValidated
methImpl_RuntimeManipulationDetailsVC_pushSuccessPage
methImpl_RuntimeManipulationDetailsVC_showLoginFailureAlert
methImpl_RuntimeManipulationDetailsVC_usernameTextField
methImpl_RuntimeManipulationDetailsVC_setUsernameTextField_
methImpl_RuntimeManipulationDetailsVC_passwordTextField
methImpl_RuntimeManipulationDetailsVC_setPasswordTextField_
methImpl_RuntimeManipulationDetailsVC__cxx_destruct

```
0x0001f00e E047          blx      r12
0x0001f010 04B0          add      sp, #0x10
0x0001f012 80BD          pop      {r7, pc}
                   ; endp
-------------------------------------------------------------------------------

================ B E G I N   O F   P R O C E D U R E ================

                          ; Basic Block Input Regs: r0 r1 r2 r4 r5 r7 sp lr pc -  Killed Regs: r
                         methImpl_ApplicationPatchingDetailsVC_jailbreakTestTapped_:
0x0001f014 B0B5          push   | {r4, r5, r7, lr}
0x0001f016 02AF          add      r7, sp, #0x8
0x0001f018 B3B0          sub      sp, #0xcc
0x0001f01a 30AB          add      r3, sp, #0xc0
0x0001f01c 40F20009      movw     r9, #0x0
0x0001f020 C0F20009      movt     r9, #0x0
0x0001f024 3290          str      r0, [sp, #0xc8]
0x0001f026 3191          str      r1, [sp, #0xc4]
0x0001f028 CDF8C090      str.w    r9, [sp, #0xc0]
0x0001f02c 1846          mov      r0, r3
0x0001f02e 1146          mov      r1, r2
0x0001f030 14F094EF      blx      imp___symbolstub1__objc_storeStrong
0x0001f034 45F2CC00      movw     r0, #0x50cc
0x0001f038 C0F20100      movt     r0, #0x1
0x0001f03c 7844          add      r0, pc                          ; 0x3410c
0x0001f03e 0068          ldr      r0, [r0]                        ; @imp___nl_symbol_
0x0001f040 4AF21C71      movw     r1, #0xa71c
0x0001f044 C0F20101      movt     r1, #0x1
0x0001f048 7944          add      r1, pc                          ; 0x39768
0x0001f04a 4AF65A22      movw     r2, #0xaa5a
0x0001f04e C0F20102      movt     r2, #0x1
0x0001f052 7A44          add      r2, pc                          ; 0x39ab0
0x0001f054 0023          movs     r3, #0x0
0x0001f056 C0F20003      movt     r3, #0x0
0x0001f05a 8DF8BC30      strb.w   r3, [sp, #0xbc]
0x0001f05e 1268          ldr      r2, [r2]                        ; @bind__OBJC_CLASS
0x0001f060 0968          ldr      r1, [r1]                        ; @selector(default
0x0001f062 2B90          str      r0, [sp, #0xac]
```

```
Analysis segment __objc_const
Analysis segment __objc_selrefs
Analysis segment __objc_classrefs
Analysis segment __objc_superrefs
Analysis segment __objc_data
Analysis segment __cfstring
Analysis segment __objc_ivar
Analysis segment __data
Analysis segment __bss
Background analysis ended
```

Address 0x1f014, Segment __text, methImpl_ApplicationPatchingDetailsVC_jailbreakTestTapped_ + 0, file offset 0x1b014

In the left side under *Label,* search for “jailbreak”. We can see the method we are concerned with in the search results (highlighted)

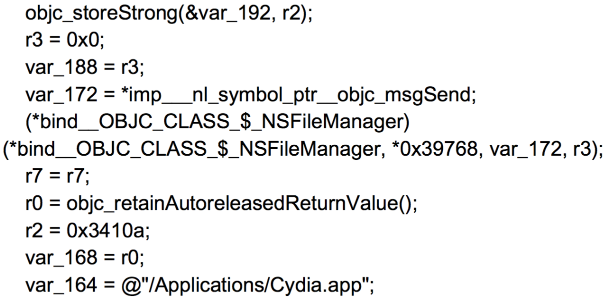


methImpl_JailbreakDetectionVC_initWithNibName_bundle_
methImpl_JailbreakDetectionVC_viewDidLoad
methImpl_JailbreakDetectionVC_didReceiveMemoryWarning
methImpl_JailbreakDetectionVC_readArticleTapped_
methImpl_JailbreakDetectionVC_jailbreakTest1Tapped_
methImpl_JailbreakDetectionVC_jailbreakTest2Tapped_
methImpl_JailbreakDetectionVC_isJailbroken
methImpl_static_DamnVulnerableAppUtilities_showAlertForJailbr…
methImpl_ApplicationPatchingDetailsVC_jailbreakTestTapped_
_OBJC_CLASS_$_JailbreakDetectionVC
_OBJC_METACLASS_$_JailbreakDetectionVC
cfstring_http___highaltitudehacks_com_2013_12_17_ios_applica…
cfstring__private_jailbreak_txt

```
0x0001f00e E047          blx      r12
0x0001f010 04B0          add      sp, #0x10
0x0001f012 80BD          pop      {r7, pc}
                   ; endp
-------------------------------------------------------------------------------

================ B E G I N   O F   P R O C E D U R E ================

                          ; Basic Block Input Regs: r0 r1 r2 r4 r5 r7 sp lr pc -  Killed Regs: r0
                         methImpl_ApplicationPatchingDetailsVC_jailbreakTestTapped_:
0x0001f014 B0B5          push   | {r4, r5, r7, lr}
0x0001f016 02AF          add      r7, sp, #0x8
0x0001f018 B3B0          sub      sp, #0xcc
0x0001f01a 30AB          add      r3, sp, #0xc0
0x0001f01c 40F20009      movw     r9, #0x0
0x0001f020 C0F20009      movt     r9, #0x0
0x0001f024 3290          str      r0, [sp, #0xc8]
0x0001f026 3191          str      r1, [sp, #0xc4]
0x0001f028 CDF8C090      str.w    r9, [sp, #0xc0]
0x0001f02c 1846          mov      r0, r3
0x0001f02e 1146          mov      r1, r2
0x0001f030 14F094EF      blx      imp___symbolstub1__objc_storeStrong
0x0001f034 45F2CC00      movw     r0, #0x50cc
0x0001f038 C0F20100      movt     r0, #0x1
0x0001f03c 7844          add      r0, pc                          ; 0x3410c
0x0001f03e 0068          ldr      r0, [r0]                        ; @imp___nl_symbol_pt
0x0001f040 4AF21C71      movw     r1, #0xa71c
0x0001f044 C0F20101      movt     r1, #0x1
0x0001f048 7944          add      r1, pc                          ; 0x39768
0x0001f04a 4AF65A22      movw     r2, #0xaa5a
0x0001f04e C0F20102      movt     r2, #0x1
0x0001f052 7A44          add      r2, pc                          ; 0x39ab0
```

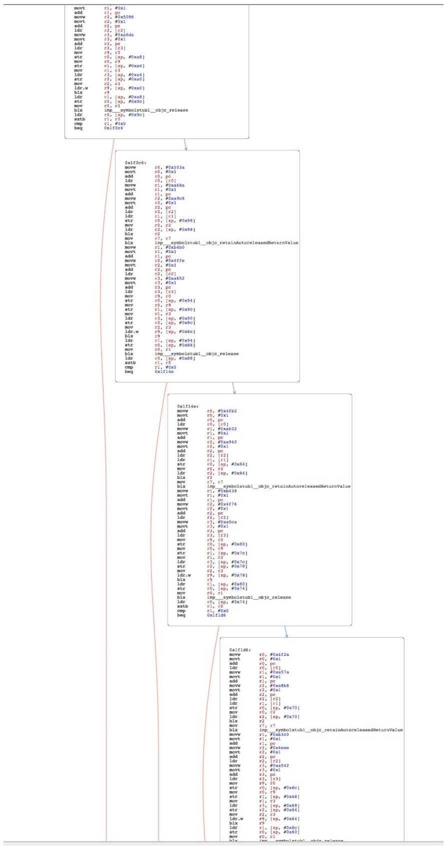
You can also see the CFG and Pseudo code for this method by tapping on *CFG* and *Pseudo code* respectively. You can find the CFG and Pseudo code for this method in the same folder.

By looking at the Pseudo code, we can clearly figure out that a lot of tests are being carried out in order to detect whether the device is jailbroken or not. For e.g, it is clear from this section of pseudo-code that the path for the Cydia application is being checked. If the Cydia application is found to be installed, then we can be sure that the device is jailbroken.

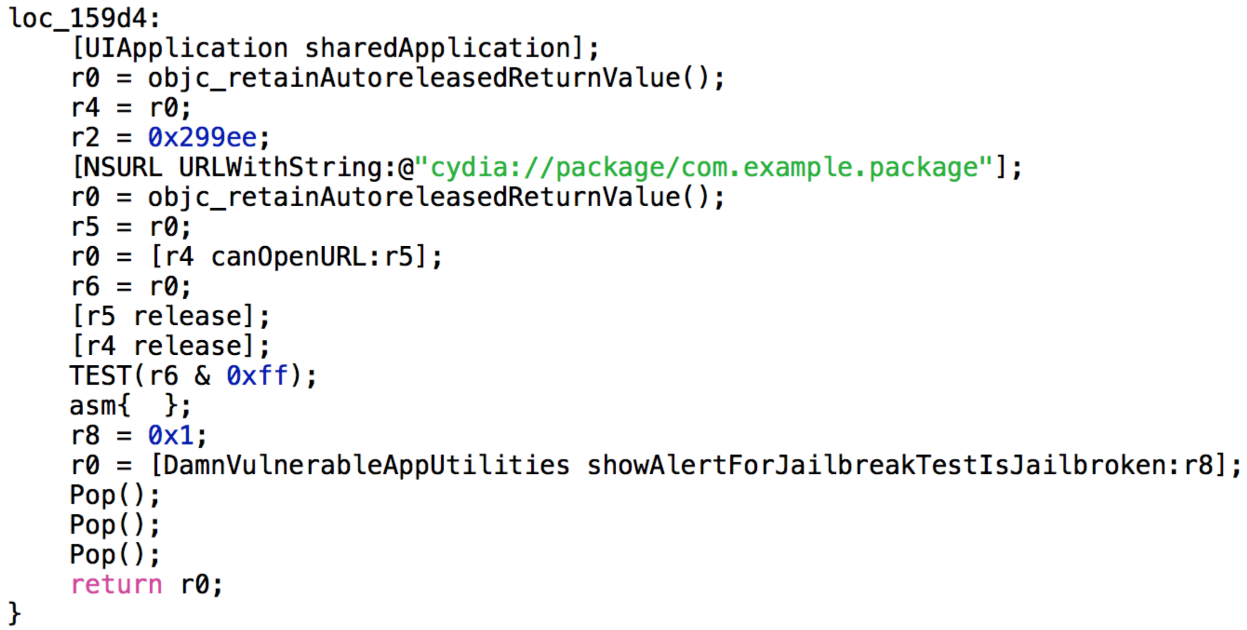


```
   objc_storeStrong(&var_192, r2);
   r3 = 0x0;
   var_188 = r3;
   var_172 = *imp___nl_symbol_ptr__objc_msgSend;
   (*bind__OBJC_CLASS_$_NSFileManager)
(*bind__OBJC_CLASS_$_NSFileManager, *0x39768, var_172, r3);
   r7 = r7;
   r0 = objc_retainAutoreleasedReturnValue();
   r2 = 0x3410a;
   var_168 = r0;
   var_164 = @"/Applications/Cydia.app";
```

From the CFG as shown below, we can also see that a number of checks are happening for jailbreak. If you have a little bit of knowledge on how jailbreak detection works, you will know that no single test can be sufficient to detect a jailbroken device, and hence multiple checks are happening in this method, but if even one test returns a positive result (device is jailbroken) then we can be sure that the device is jailbroken. If on the other hand a test returns a negative result (device is not jailbroken) then it would be wrong to assume that the device is not jailbroken by just having your conclusions on that single test.



If we look at the Pseudo-code again, we can see this section at the very bottom.



```
loc_159d4:
    [UIApplication sharedApplication];
    r0 = objc_retainAutoreleasedReturnValue();
    r4 = r0;
    r2 = 0x299ee;
    [NSURL URLWithString:@"cydia://package/com.example.package"];
    r0 = objc_retainAutoreleasedReturnValue();
    r5 = r0;
    r0 = [r4 canOpenURL:r5];
    r6 = r0;
    [r5 release];
    [r4 release];
    TEST(r6 & 0xff);
    asm{  };
    r8 = 0x1;
    r0 = [DamnVulnerableAppUtilities showAlertForJailbreakTestIsJailbroken:r8];
    Pop();
    Pop();
    Pop();
    return r0;
}
```

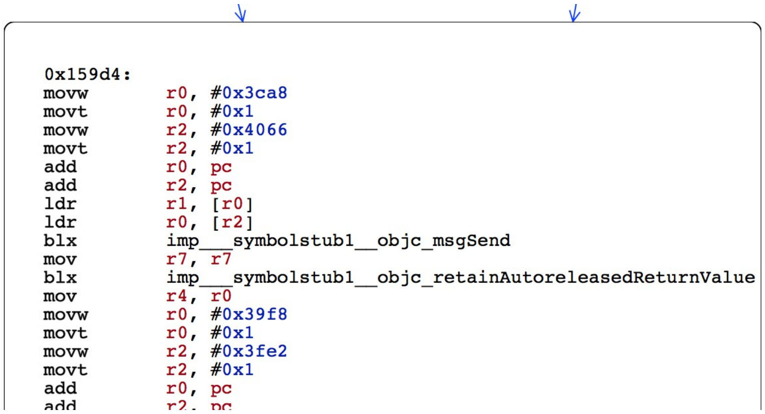
The two lines that are of interest here are

r8 = 0x1;

*r0 = [DamnVulnerableAppUtilities showAlertForJailbreakTestIsJailbroken:r8]***;**

So the value of the register r8 is set to 1 and then the method *[DamnVulnerableAppUtilities showAlertForJailbreakTestIsJailbroken:r8]* is called with the argument *r8* as 1. It looks like this method takes a boolean parameter and shows the alert for jailbreak depending on that boolean value. Since the argument here is 1, we can assume that the flow will go to this section (with label *loc_159d4*) only if it has been decided that the device is jailbroken. Since our task is to show an alert which states that the device is not jailbroken, we can do the following things.

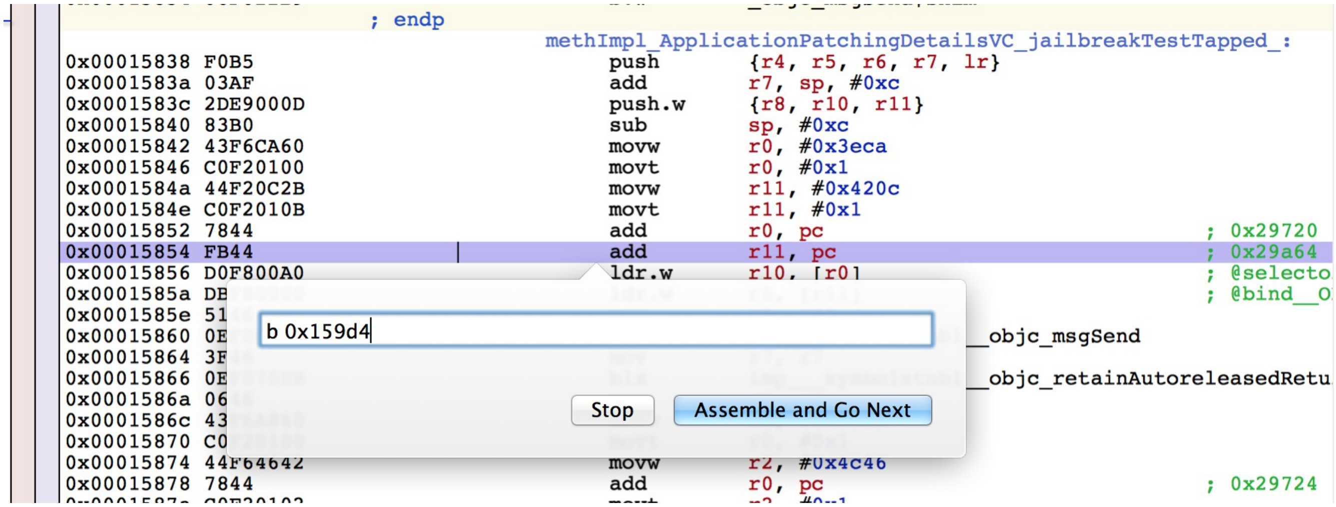
a) Make the flow go to this section of code by calling a branch instruction from anywhere in the code. From the CFG, we can see that the label for the section of code where we want the flow to reach is *0x159d4*



```
0x159d4:
movw        r0, #0x3ca8
movt        r0, #0x1
movw        r2, #0x4066
movt        r2, #0x1
add         r0, pc
add         r2, pc
ldr         r1, [r0]
ldr         r0, [r2]
blx         imp___symbolstub1__objc_msgSend
mov         r7, r7
blx         imp___symbolstub1__objc_retainAutoreleasedReturnValue
mov         r4, r0
movw        r0, #0x39f8
movt        r0, #0x1
movw        r2, #0x3fe2
movt        r2, #0x1
add         r0, pc
add         r2, pc
```

b) Set the value of r8 to 0 instead of 1 before it is passed to the method *jailbreakTestTapped* as an argument.

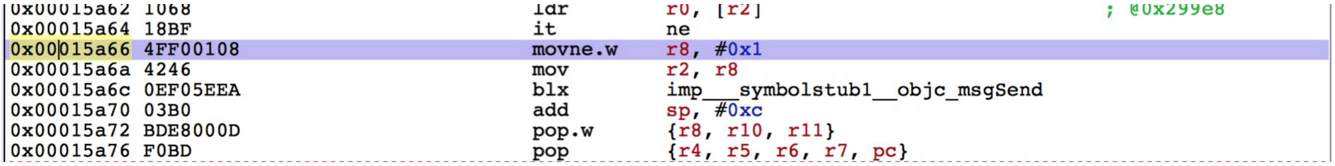
Let's go to Hopper and in the disassembly for the method we are concerned with (*jailbreakTestTapped*), click on any instruction from the beginning (make sure the flow reaches this instruction) and go to *Modify → Assemble Instruction.* Then we write a branch instruction to our section of code, the instruction will be *b 0x159d4* where b stands for the branch instruction in ARM assembly.



```
                                 ; endp
                                        methImpl_ApplicationPatchingDetailsVC_jailbreakTestTapped_:
0x00015838 F0B5                         push        {r4, r5, r6, r7, lr}
0x0001583a 03AF                         add         r7, sp, #0xc
0x0001583c 2DE9000D                     push.w      {r8, r10, r11}
0x00015840 83B0                         sub         sp, #0xc
0x00015842 43F6CA60                     movw        r0, #0x3eca
0x00015846 C0F20100                     movt        r0, #0x1
0x0001584a 44F20C2B                     movw        r11, #0x420c
0x0001584e C0F2010B                     movt        r11, #0x1
0x00015852 7844                         add         r0, pc                   ; 0x29720
0x00015854 FB44                         add         r11, pc                  ; 0x29a64
0x00015856 D0F800A0                     ldr.w       r10, [r0]                ; @selecto
0x0001585a DB                                                                ; @bind__O
0x0001585e 51
0x00015860 0E                                   _objc_msgSend
0x00015864 3F
0x00015866 0E                                   _objc_retainAutoreleasedRetu
0x0001586a 06
0x0001586c 43
0x00015870 C0
0x00015874 44F64642                     movw        r2, #0x4c46
0x00015878 7844                         add         r0, pc                   ; 0x29724
```

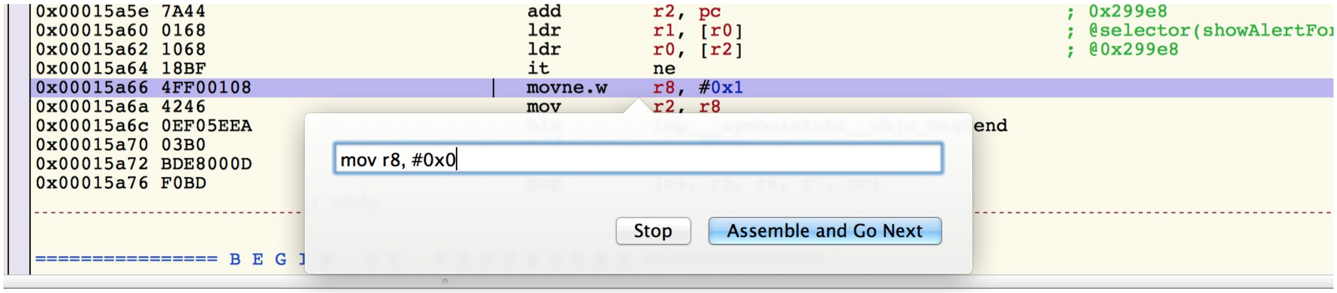
b 0x159d4

Then let's head over to the particular section of code with the label 0x159d4 in the disassembly and look for the section of code where the value 1 is being moved to the register r8.



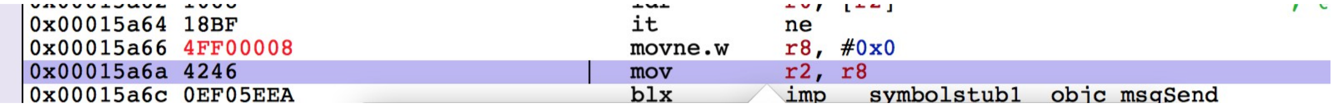
```
0x00015a62 1068                 ldr         r0, [r2]                  ; @0x299e8
0x00015a64 18BF                 it          ne
0x00015a66 4FF00108             movne.w     r8, #0x1
0x00015a6a 4246                 mov         r2, r8
0x00015a6c 0EF05EEA             blx         imp___symbolstub1__objc_msgSend
0x00015a70 03B0                 add         sp, #0xc
0x00015a72 BDE8000D             pop.w       {r8, r10, r11}
0x00015a76 F0BD                 pop         {r4, r5, r6, r7, pc}
```

And modify this instruction to instead pass 0 to the r8 register.



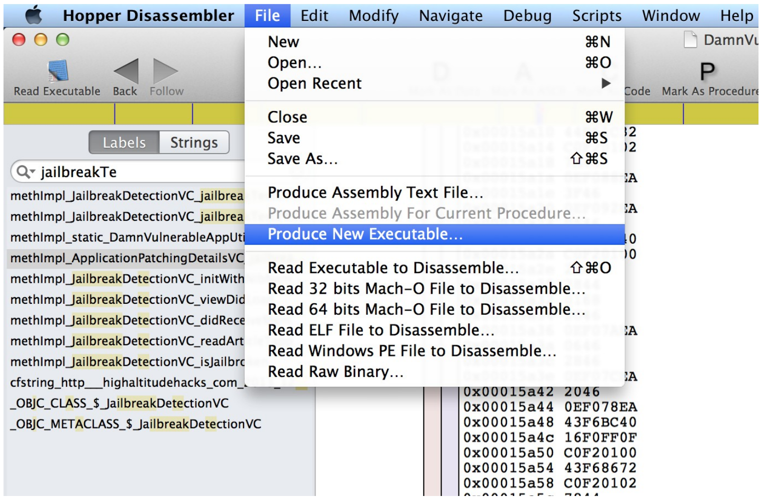
```
0x00015a5e 7A44                add     r2, pc               ; 0x299e8
0x00015a60 0168                ldr     r1, [r0]             ; @selector(showAlertFo
0x00015a62 1068                ldr     r0, [r2]             ; @0x299e8
0x00015a64 18BF                it      ne
0x00015a66 4FF00108            movne.w r8, #0x1
0x00015a6a 4246                mov     r2, r8
0x00015a6c 0EF05EEA                                                 end
0x00015a70 03B0
0x00015a72 BDE8000D
0x00015a76 F0BD

================ B E G I
```

After the change, this is how the disassembly will look like.



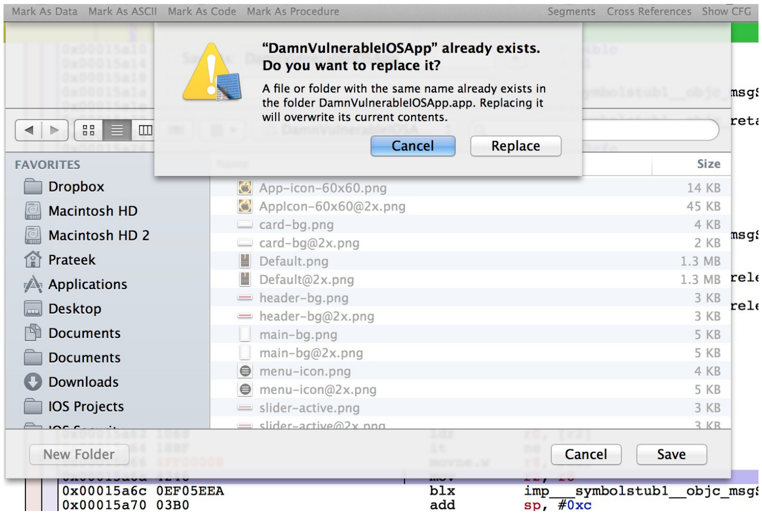
```
0x00015a64 18BF       it       ne
0x00015a66 4FF00008   movne.w  r8, #0x0
0x00015a6a 4246       mov      r2, r8
0x00015a6c 0EF05EEA   blx      imp    symbolstub1   objc msgSend
```

Ok, so our binary has now been modified. Save it and overwrite the previous executable.



Hopper Disassembler
File
Edit
Modify
Navigate
Debug
Scripts
Window
Help
Read Executable
Back
Follow
Code
Mark As Procedure
DamnVu
Labels
Strings
jailbreakTe
methImpl_JailbreakDetectionVC_jailbrea
methImpl_JailbreakDetectionVC_jailbrea
methImpl_static_DamnVulnerableAppUti
methImpl_ApplicationPatchingDetailsVC
methImpl_JailbreakDetectionVC_initWith
methImpl_JailbreakDetectionVC_viewDid
methImpl_JailbreakDetectionVC_didRece
methImpl_JailbreakDetectionVC_readArt
methImpl_JailbreakDetectionVC_isJailbr
cfstring_http__highaltitudehacks_com_
_OBJC_CLASS_$_JailbreakDetectionVC
_OBJC_METACLASS_$_JailbreakDetectionVC
New ⌘N
Open… ⌘O
Open Recent ▶
Close ⌘W
Save ⌘S
Save As… ⇧⌘S
Produce Assembly Text File…
Produce Assembly For Current Procedure…
Produce New Executable…
Read Executable to Disassemble… ⇧⌘O
Read 32 bits Mach-O File to Disassemble…
Read 64 bits Mach-O File to Disassemble…
Read ELF File to Disassemble…
Read Windows PE File to Disassemble…
Read Raw Binary…

```
0x00015a42 2046
0x00015a44 0EF078EA
0x00015a48 43F6BC40
0x00015a4c 16F0FF0F
0x00015a50 C0F20100
0x00015a54 43F68672
0x00015a58 C0F20102
```



"DamnVulnerableIOSApp" already exists. Do you want to replace it?

A file or folder with the same name already exists in the folder DamnVulnerableIOSApp.app. Replacing it will overwrite its current contents.

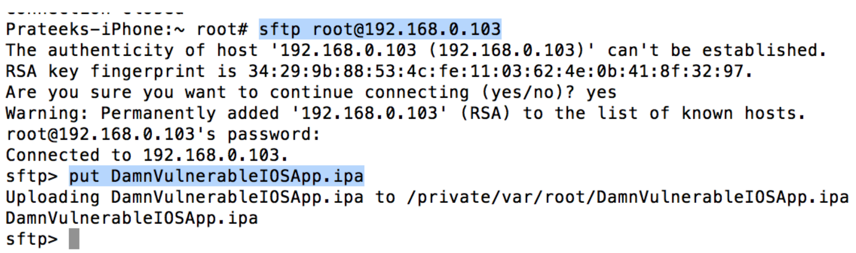
	Size
App-icon-60x60.png	14 KB
AppIcon-60x60@2x.png	45 KB
card-bg.png	4 KB
card-bg@2x.png	2 KB
Default.png	1.3 MB
Default@2x.png	1.3 MB
header-bg.png	3 KB
header-bg@2x.png	3 KB
main-bg.png	5 KB
main-bg@2x.png	5 KB
menu-icon.png	4 KB
menu-icon@2x.png	5 KB
slider-active.png	3 KB
slider-active@2x.png	3 KB

```
0x00015a6c 0EF05EEA    blx    imp___symbolstub1__objc_msg
0x00015a70 03B0        add    sp, #0xc
```

Now create a folder named Payload, put the DamnVulnerableIOSApp.app file under it (note that it will have the new binary now), compress that folder (it will be initially named as Payload.zip) and name it

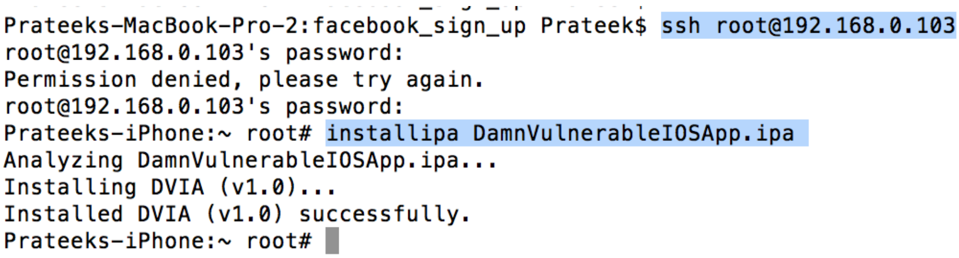
DamnVulnerableIOSApp.ipa .

Sftp to your device and upload this ipa file.



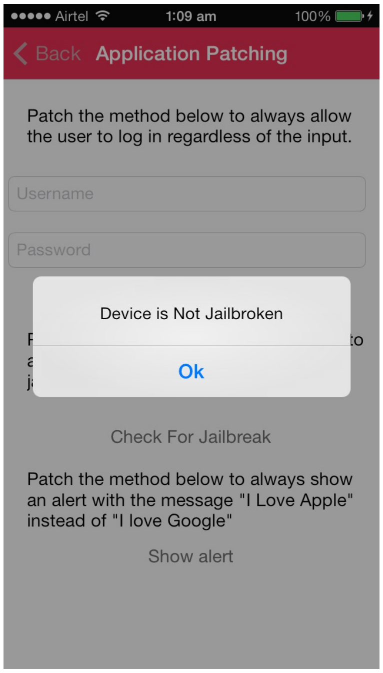
```
Prateeks-iPhone:~ root# sftp root@192.168.0.103
The authenticity of host '192.168.0.103 (192.168.0.103)' can't be established.
RSA key fingerprint is 34:29:9b:88:53:4c:fe:11:03:62:4e:0b:41:8f:32:97.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.0.103' (RSA) to the list of known hosts.
root@192.168.0.103's password:
Connected to 192.168.0.103.
sftp> put DamnVulnerableIOSApp.ipa
Uploading DamnVulnerableIOSApp.ipa to /private/var/root/DamnVulnerableIOSApp.ipa
DamnVulnerableIOSApp.ipa
sftp> 
```

Now ssh into your device and install the DVIA application using the command line utility *installipa.* Make sure that you have the utility AppSync already installed on your jailbroken device or this installation might fail.



```
Prateeks-MacBook-Pro-2:facebook_sign_up Prateek$ ssh root@192.168.0.103
root@192.168.0.103's password:
Permission denied, please try again.
root@192.168.0.103's password:
Prateeks-iPhone:~ root# installipa DamnVulnerableIOSApp.ipa
Analyzing DamnVulnerableIOSApp.ipa...
Installing DVIA (v1.0)...
Installed DVIA (v1.0) successfully.
Prateeks-iPhone:~ root# 
```

Now if you go to the Binary patching section in the app and tap on *Check For Jailbreak,* we can see that the check fails even though we are running the application on a jailbroken device.



Patch the method below to always allow the user to log in regardless of the input.

Device is Not Jailbroken

Patch the method below to always show an alert with the message "I Love Apple" instead of "I love Google"