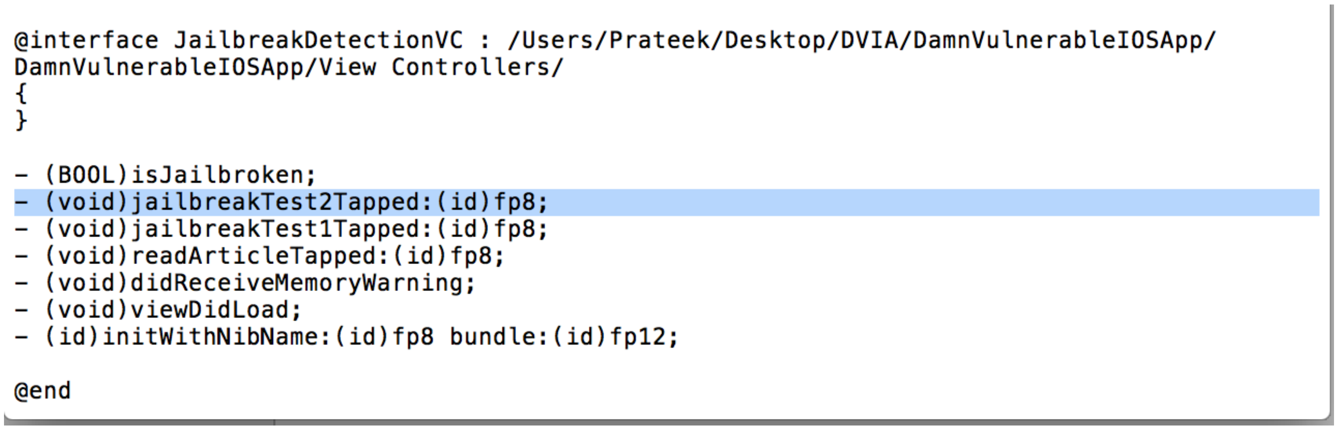
# Damn Vulnerable IOS Application Solutions

**http://damnvulnerableiosapp.com/**

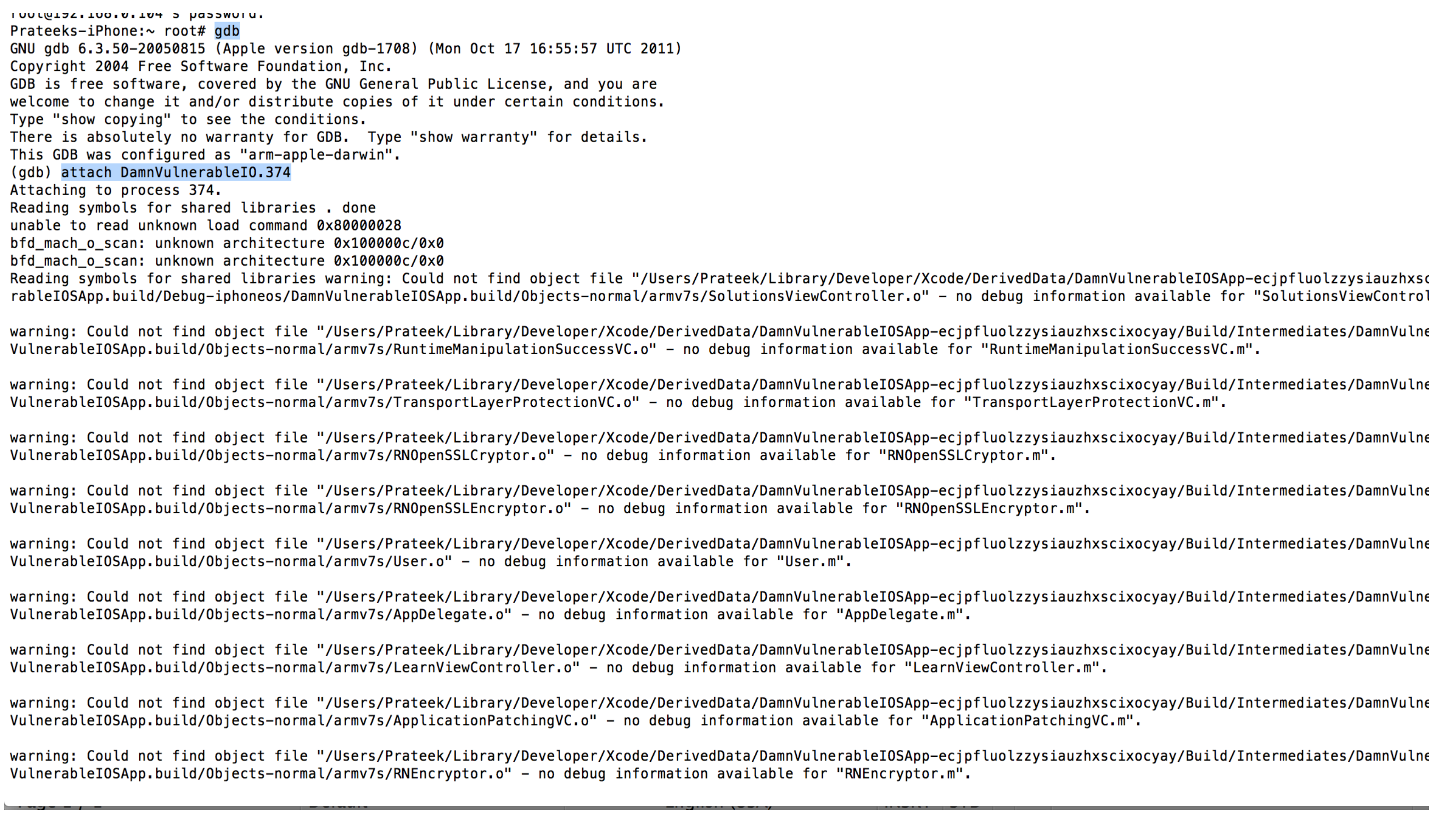
## Jailbreak Detection – Jailbreak Test 2

By now you must have figured out that the solution for Jailbreak Test 1 doesn't work on Jailbreak Test 2. So things are a bit complicated here. Let's look at the class information for this application. You can find the class information for this application in the folder *Other Files* or you can just use class-dump to dump out the information. After scrolling down a bit, we can see the method that gets called on tapping the button *Jailbreak Test 2*

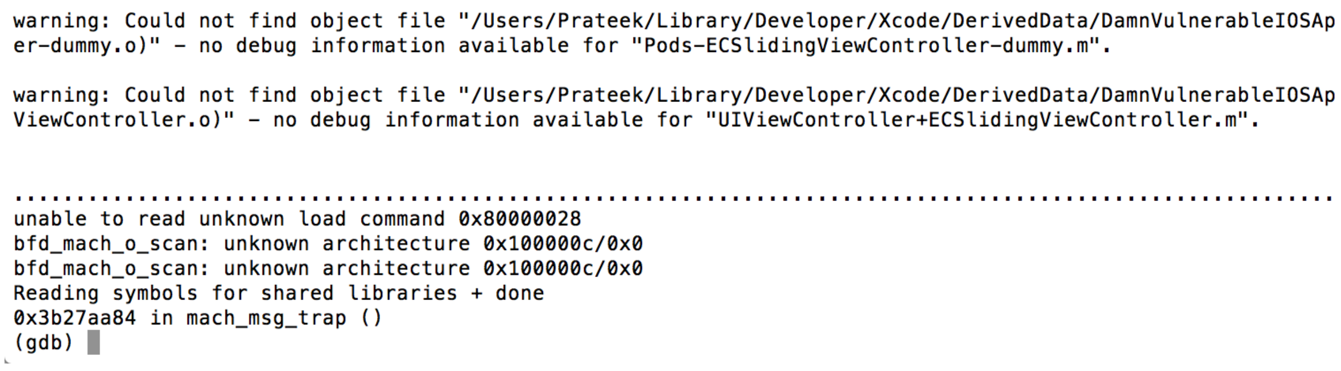


```
@interface JailbreakDetectionVC : /Users/Prateek/Desktop/DVIA/DamnVulnerableIOSApp/
DamnVulnerableIOSApp/View Controllers/
{
}

- (BOOL)isJailbroken;
- (void)jailbreakTest2Tapped:(id)fp8;
- (void)jailbreakTest1Tapped:(id)fp8;
- (void)readArticleTapped:(id)fp8;
- (void)didReceiveMemoryWarning;
- (void)viewDidLoad;
- (id)initWithNibName:(id)fp8 bundle:(id)fp12;

@end
```

Let's start GDB on our device. After starting GDB, make sure the application is running on foreground and attach to to it.

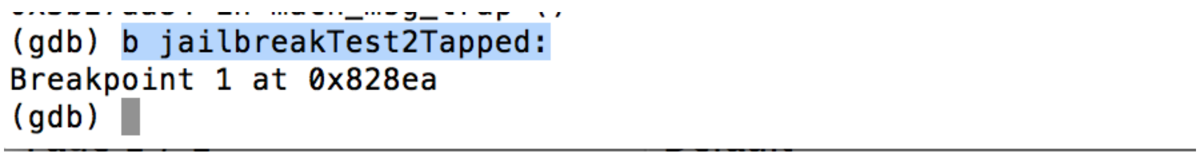


```
root@192.168.0.104's password:
Prateeks-iPhone:~ root# gdb
GNU gdb 6.3.50-20050815 (Apple version gdb-1708) (Mon Oct 17 16:55:57 UTC 2011)
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "arm-apple-darwin".
(gdb) attach DamnVulnerableIO.374
Attaching to process 374.
Reading symbols for shared libraries . done
unable to read unknown load command 0x80000028
bfd_mach_o_scan: unknown architecture 0x100000c/0x0
bfd_mach_o_scan: unknown architecture 0x100000c/0x0
Reading symbols for shared libraries warning: Could not find object file "/Users/Prateek/Library/Developer/Xcode/DerivedData/DamnVulnerableIOSApp-ecjpfluolzzysiauzhxsc
rableIOSApp.build/Debug-iphoneos/DamnVulnerableIOSApp.build/Objects-normal/armv7s/SolutionsViewController.o" - no debug information available for "SolutionsViewControl

warning: Could not find object file "/Users/Prateek/Library/Developer/Xcode/DerivedData/DamnVulnerableIOSApp-ecjpfluolzzysiauzhxscixocyay/Build/Intermediates/DamnVulne
VulnerableIOSApp.build/Objects-normal/armv7s/RuntimeManipulationSuccessVC.o" - no debug information available for "RuntimeManipulationSuccessVC.m".

warning: Could not find object file "/Users/Prateek/Library/Developer/Xcode/DerivedData/DamnVulnerableIOSApp-ecjpfluolzzysiauzhxscixocyay/Build/Intermediates/DamnVulne
VulnerableIOSApp.build/Objects-normal/armv7s/TransportLayerProtectionVC.o" - no debug information available for "TransportLayerProtectionVC.m".

warning: Could not find object file "/Users/Prateek/Library/Developer/Xcode/DerivedData/DamnVulnerableIOSApp-ecjpfluolzzysiauzhxscixocyay/Build/Intermediates/DamnVulne
VulnerableIOSApp.build/Objects-normal/armv7s/RNOpenSSLCryptor.o" - no debug information available for "RNOpenSSLCryptor.m".

warning: Could not find object file "/Users/Prateek/Library/Developer/Xcode/DerivedData/DamnVulnerableIOSApp-ecjpfluolzzysiauzhxscixocyay/Build/Intermediates/DamnVulne
VulnerableIOSApp.build/Objects-normal/armv7s/RNOpenSSLEncryptor.o" - no debug information available for "RNOpenSSLEncryptor.m".

warning: Could not find object file "/Users/Prateek/Library/Developer/Xcode/DerivedData/DamnVulnerableIOSApp-ecjpfluolzzysiauzhxscixocyay/Build/Intermediates/DamnVulne
VulnerableIOSApp.build/Objects-normal/armv7s/User.o" - no debug information available for "User.m".

warning: Could not find object file "/Users/Prateek/Library/Developer/Xcode/DerivedData/DamnVulnerableIOSApp-ecjpfluolzzysiauzhxscixocyay/Build/Intermediates/DamnVulne
VulnerableIOSApp.build/Objects-normal/armv7s/AppDelegate.o" - no debug information available for "AppDelegate.m".

warning: Could not find object file "/Users/Prateek/Library/Developer/Xcode/DerivedData/DamnVulnerableIOSApp-ecjpfluolzzysiauzhxscixocyay/Build/Intermediates/DamnVulne
VulnerableIOSApp.build/Objects-normal/armv7s/LearnViewController.o" - no debug information available for "LearnViewController.m".

warning: Could not find object file "/Users/Prateek/Library/Developer/Xcode/DerivedData/DamnVulnerableIOSApp-ecjpfluolzzysiauzhxscixocyay/Build/Intermediates/DamnVulne
VulnerableIOSApp.build/Objects-normal/armv7s/ApplicationPatchingVC.o" - no debug information available for "ApplicationPatchingVC.m".

warning: Could not find object file "/Users/Prateek/Library/Developer/Xcode/DerivedData/DamnVulnerableIOSApp-ecjpfluolzzysiauzhxscixocyay/Build/Intermediates/DamnVulne
VulnerableIOSApp.build/Objects-normal/armv7s/RNEncryptor.o" - no debug information available for "RNEncryptor.m".
```



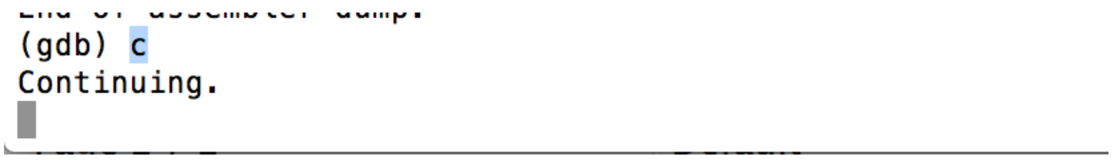
```
warning: Could not find object file "/Users/Prateek/Library/Developer/Xcode/DerivedData/DamnVulnerableIOSAp
er-dummy.o)" - no debug information available for "Pods-ECSlidingViewController-dummy.m".

warning: Could not find object file "/Users/Prateek/Library/Developer/Xcode/DerivedData/DamnVulnerableIOSAp
ViewController.o)" - no debug information available for "UIViewController+ECSlidingViewController.m".


.........................................................................................................
unable to read unknown load command 0x80000028
bfd_mach_o_scan: unknown architecture 0x100000c/0x0
bfd_mach_o_scan: unknown architecture 0x100000c/0x0
Reading symbols for shared libraries + done
0x3b27aa84 in mach_msg_trap ()
(gdb)
```

Then let's set a breakpoint for the method that gets called on tapping the button *Jailbreak Test 2.* You can set it by using the command *b jailbreakTest2Tapped:*



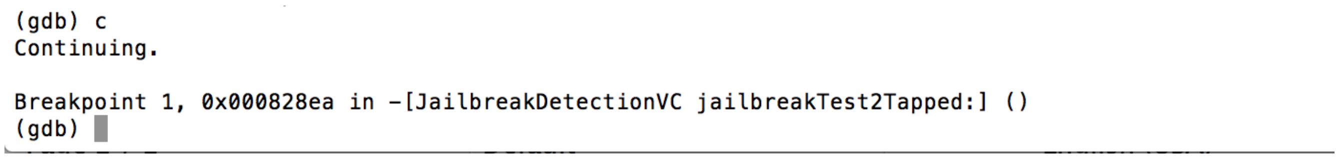
```
(gdb) b jailbreakTest2Tapped:
Breakpoint 1 at 0x828ea
(gdb)
```

Now type the **c** command to continue the application. You will see that the application on the device will now begin responsive.



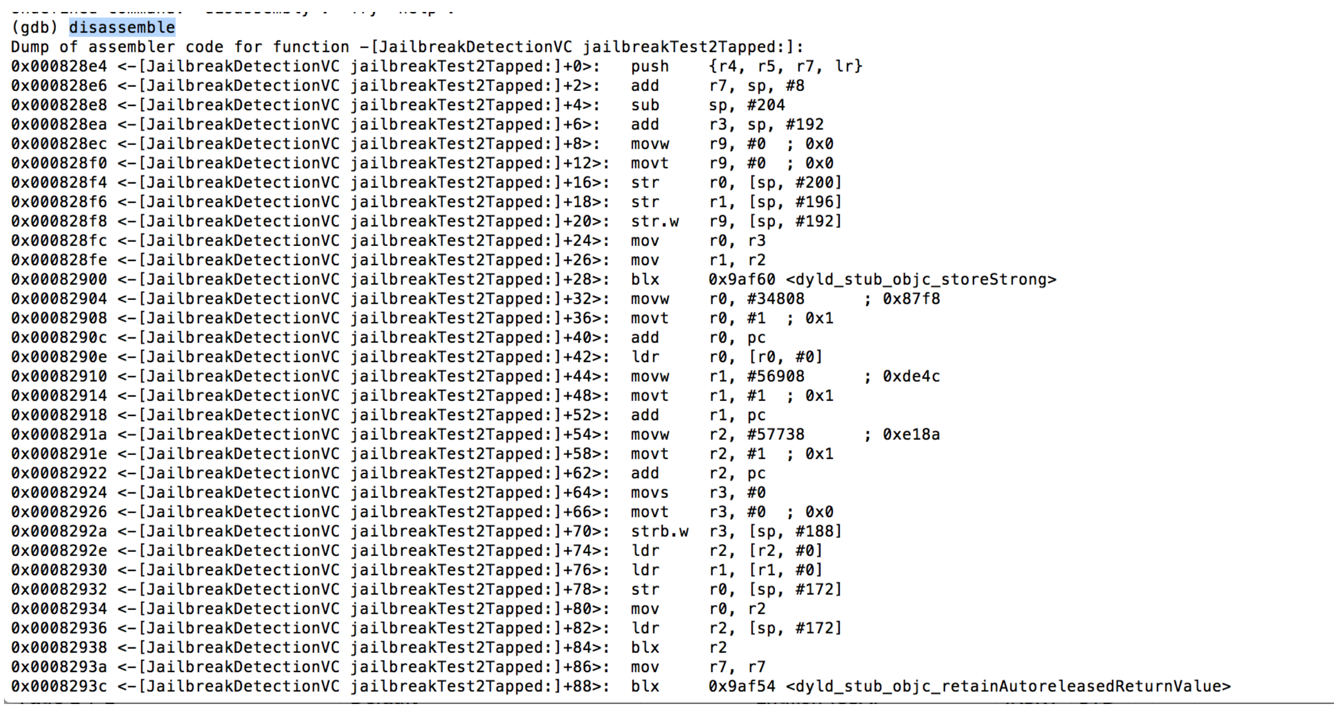
```
(gdb) c
Continuing.
```

Now let's go ahead and tap on *Jailbreak Test 2.* We will see our breakpoint being hit.



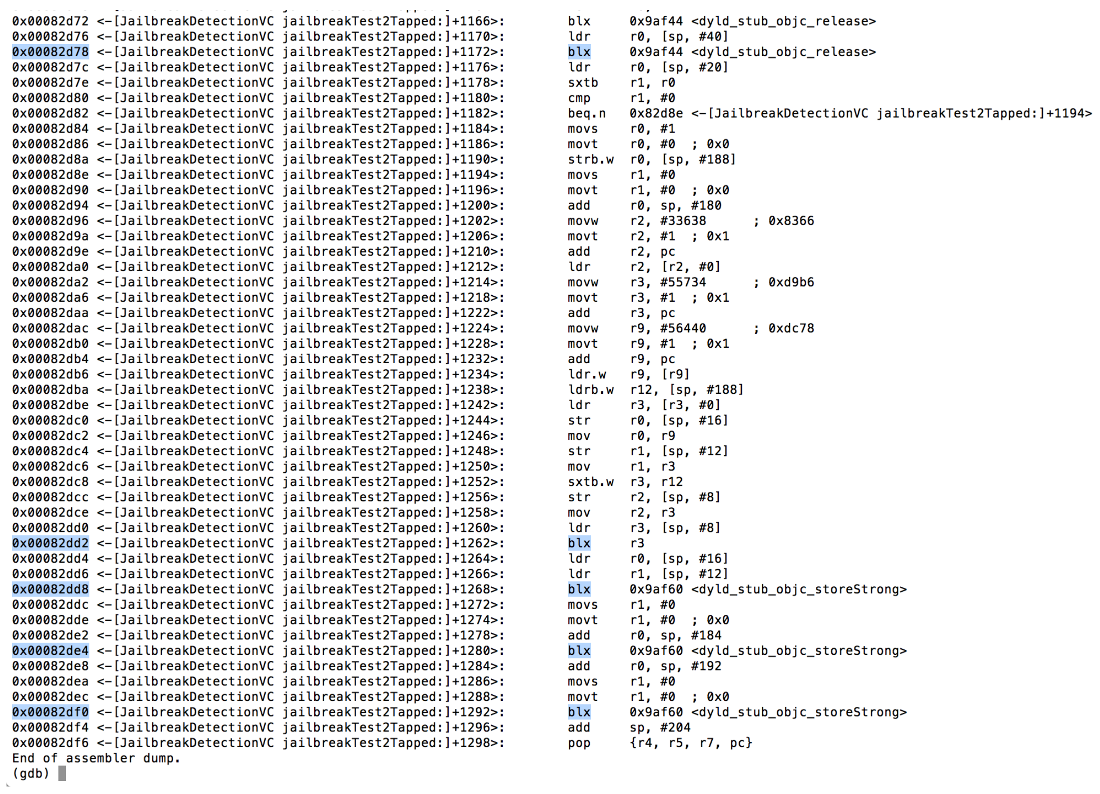
```
(gdb) c
Continuing.

Breakpoint 1, 0x000828ea in -[JailbreakDetectionVC jailbreakTest2Tapped:] ()
(gdb)
```

Let's disassemble the code. Use the *disassemble* command to see the disassembly.


```
(gdb) disassemble
Dump of assembler code for function -[JailbreakDetectionVC jailbreakTest2Tapped:]:
0x000828e4 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+0>:   push    {r4, r5, r7, lr}
0x000828e6 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+2>:   add     r7, sp, #8
0x000828e8 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+4>:   sub     sp, #204
0x000828ea <-[JailbreakDetectionVC jailbreakTest2Tapped:]+6>:   add     r3, sp, #192
0x000828ec <-[JailbreakDetectionVC jailbreakTest2Tapped:]+8>:   movw    r9, #0  ; 0x0
0x000828f0 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+12>:  movt    r9, #0  ; 0x0
0x000828f4 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+16>:  str     r0, [sp, #200]
0x000828f6 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+18>:  str     r1, [sp, #196]
0x000828f8 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+20>:  str.w   r9, [sp, #192]
0x000828fc <-[JailbreakDetectionVC jailbreakTest2Tapped:]+24>:  mov     r0, r3
0x000828fe <-[JailbreakDetectionVC jailbreakTest2Tapped:]+26>:  mov     r1, r2
0x00082900 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+28>:  blx     0x9af60 <dyld_stub_objc_storeStrong>
0x00082904 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+32>:  movw    r0, #34808      ; 0x87f8
0x00082908 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+36>:  movt    r0, #1  ; 0x1
0x0008290c <-[JailbreakDetectionVC jailbreakTest2Tapped:]+40>:  add     r0, pc
0x0008290e <-[JailbreakDetectionVC jailbreakTest2Tapped:]+42>:  ldr     r0, [r0, #0]
0x00082910 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+44>:  movw    r1, #56908      ; 0xde4c
0x00082914 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+48>:  movt    r1, #1  ; 0x1
0x00082918 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+52>:  add     r1, pc
0x0008291a <-[JailbreakDetectionVC jailbreakTest2Tapped:]+54>:  movw    r2, #57738      ; 0xe18a
0x0008291e <-[JailbreakDetectionVC jailbreakTest2Tapped:]+58>:  movt    r2, #1  ; 0x1
0x00082922 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+62>:  add     r2, pc
0x00082924 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+64>:  movs    r3, #0
0x00082926 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+66>:  movt    r3, #0  ; 0x0
0x0008292a <-[JailbreakDetectionVC jailbreakTest2Tapped:]+70>:  strb.w  r3, [sp, #188]
0x0008292e <-[JailbreakDetectionVC jailbreakTest2Tapped:]+74>:  ldr     r2, [r2, #0]
0x00082930 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+76>:  ldr     r1, [r1, #0]
0x00082932 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+78>:  str     r0, [sp, #172]
0x00082934 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+80>:  mov     r0, r2
0x00082936 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+82>:  ldr     r2, [sp, #172]
0x00082938 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+84>:  blx     r2
0x0008293a <-[JailbreakDetectionVC jailbreakTest2Tapped:]+86>:  mov     r7, r7
0x0008293c <-[JailbreakDetectionVC jailbreakTest2Tapped:]+88>:  blx     0x9af54 <dyld_stub_objc_retainAutoreleasedReturnValue>
```

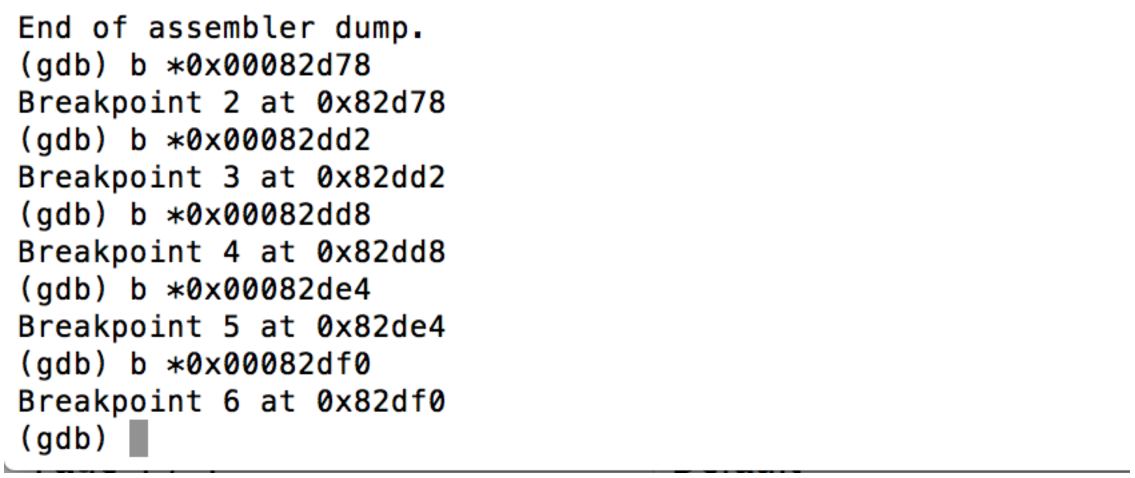
We know that whenever an external method is called or a property is accessed, the *objc_msgSend* function is called. But there are thousands of *objc_msgSend* calls called in any application. We should only be concerned with the *objc_msgSend* calls related to this function. So we find out the addresses of all the instructions that call *objc_msgSend* and set a breakpoint for it. A very simple way to do it is to look for the *blx* instruction, note its address and set a breakpoint for it. A nice trick here would be to note the *blx* instructions from the bottom, note their address, set a breakpoint for them and move upward. This is because we can safely assume that the validation will end somewhere at the end of the function and hence starting from the bottom would make more sense. This is however not guaranteed but let's just do it as it may prove to be a more efficient way.

In this case, let me select the last 5 *blx* instructions



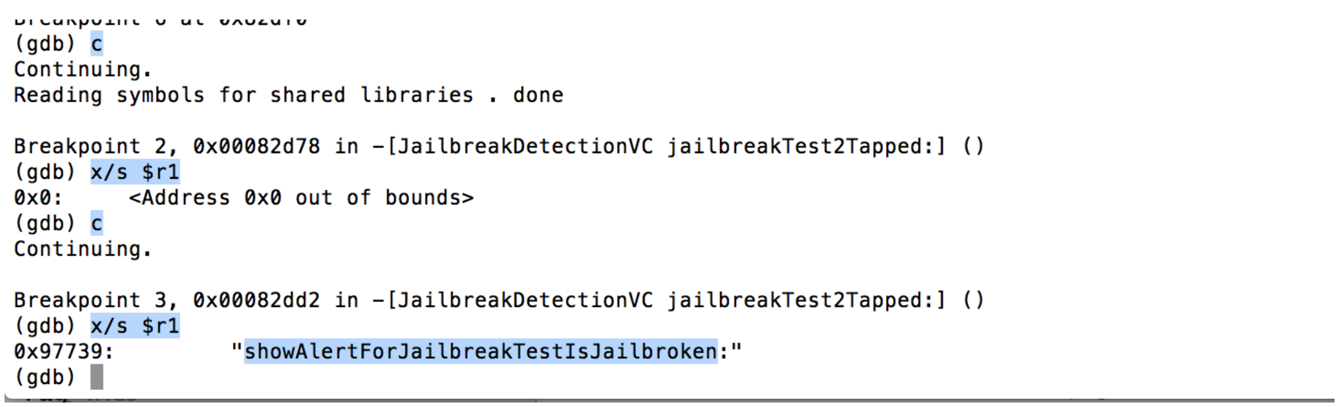
```
0x00082d72 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1166>:        blx     0x9af44 <dyld_stub_objc_release>
0x00082d76 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1170>:        ldr     r0, [sp, #40]
0x00082d78 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1172>:        blx     0x9af44 <dyld_stub_objc_release>
0x00082d7c <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1176>:        ldr     r0, [sp, #20]
0x00082d7e <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1178>:        sxtb    r1, r0
0x00082d80 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1180>:        cmp     r1, #0
0x00082d82 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1182>:        beq.n   0x82d8e <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1194>
0x00082d84 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1184>:        movs    r0, #1
0x00082d86 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1186>:        movt    r0, #0  ; 0x0
0x00082d8a <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1190>:        strb.w  r0, [sp, #188]
0x00082d8e <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1194>:        movs    r1, #0
0x00082d90 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1196>:        movt    r1, #0  ; 0x0
0x00082d94 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1200>:        add     r0, sp, #180
0x00082d96 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1202>:        movw    r2, #33638      ; 0x8366
0x00082d9a <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1206>:        movt    r2, #1  ; 0x1
0x00082d9e <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1210>:        add     r2, pc
0x00082da0 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1212>:        ldr     r2, [r2, #0]
0x00082da2 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1214>:        movw    r3, #55734      ; 0xd9b6
0x00082da6 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1218>:        movt    r3, #1  ; 0x1
0x00082daa <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1222>:        add     r3, pc
0x00082dac <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1224>:        movw    r9, #56440      ; 0xdc78
0x00082db0 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1228>:        movt    r9, #1  ; 0x1
0x00082db4 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1232>:        add     r9, pc
0x00082db6 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1234>:        ldr.w   r9, [r9]
0x00082dba <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1238>:        ldrb.w  r12, [sp, #188]
0x00082dbe <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1242>:        ldr     r3, [r3, #0]
0x00082dc0 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1244>:        str     r0, [sp, #16]
0x00082dc2 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1246>:        mov     r0, r9
0x00082dc4 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1248>:        str     r1, [sp, #12]
0x00082dc6 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1250>:        mov     r1, r3
0x00082dc8 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1252>:        sxtb.w  r3, r12
0x00082dcc <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1256>:        str     r2, [sp, #8]
0x00082dce <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1258>:        mov     r2, r3
0x00082dd0 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1260>:        ldr     r3, [sp, #8]
0x00082dd2 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1262>:        blx     r3
0x00082dd4 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1264>:        ldr     r0, [sp, #16]
0x00082dd6 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1266>:        ldr     r1, [sp, #12]
0x00082dd8 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1268>:        blx     0x9af60 <dyld_stub_objc_storeStrong>
0x00082ddc <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1272>:        movs    r1, #0
0x00082dde <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1274>:        movt    r1, #0  ; 0x0
0x00082de2 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1278>:        add     r0, sp, #184
0x00082de4 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1280>:        blx     0x9af60 <dyld_stub_objc_storeStrong>
0x00082de8 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1284>:        add     r0, sp, #192
0x00082dea <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1286>:        movs    r1, #0
0x00082dec <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1288>:        movt    r1, #0  ; 0x0
0x00082df0 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1292>:        blx     0x9af60 <dyld_stub_objc_storeStrong>
0x00082df4 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1296>:        add     sp, #204
0x00082df6 <-[JailbreakDetectionVC jailbreakTest2Tapped:]+1298>:        pop     {r4, r5, r7, pc}
End of assembler dump.
(gdb)
```

And set breakpoints for it.



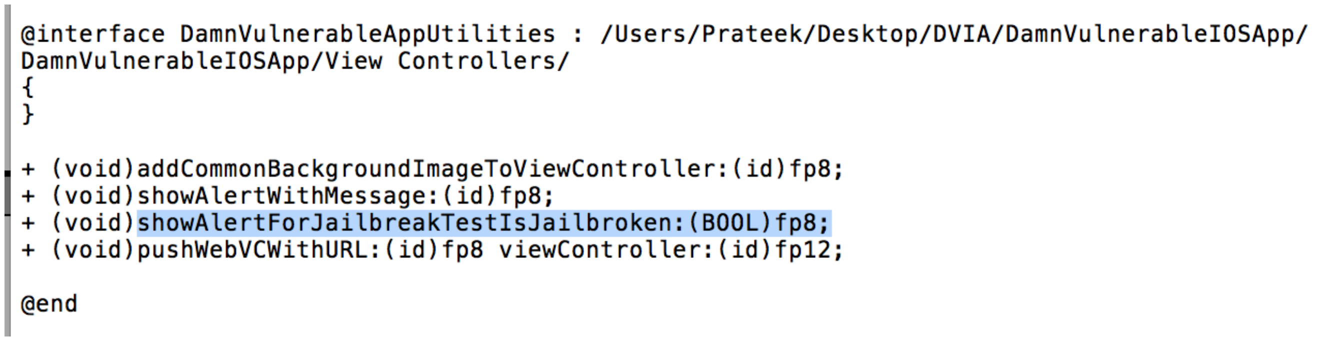
```
End of assembler dump.
(gdb) b *0x00082d78
Breakpoint 2 at 0x82d78
(gdb) b *0x00082dd2
Breakpoint 3 at 0x82dd2
(gdb) b *0x00082dd8
Breakpoint 4 at 0x82dd8
(gdb) b *0x00082de4
Breakpoint 5 at 0x82de4
(gdb) b *0x00082df0
Breakpoint 6 at 0x82df0
(gdb)
```

Now let me continue by using the **c** command . As we move through every *objc_msgSend* instruction one by one, we will print out the registers and see if there is anything of interest. We are printing out the value of *r1* register with every *objc_msgSend* call here. Then if there is nothing of interest, we just type **c** to continue until the next breakpoint is hit.



```
(gdb) c
Continuing.
Reading symbols for shared libraries . done

Breakpoint 2, 0x00082d78 in -[JailbreakDetectionVC jailbreakTest2Tapped:] ()
(gdb) x/s $r1
0x0:     <Address 0x0 out of bounds>
(gdb) c
Continuing.

Breakpoint 3, 0x00082dd2 in -[JailbreakDetectionVC jailbreakTest2Tapped:] ()
(gdb) x/s $r1
0x97739:         "showAlertForJailbreakTestIsJailbroken:"
(gdb) 
```

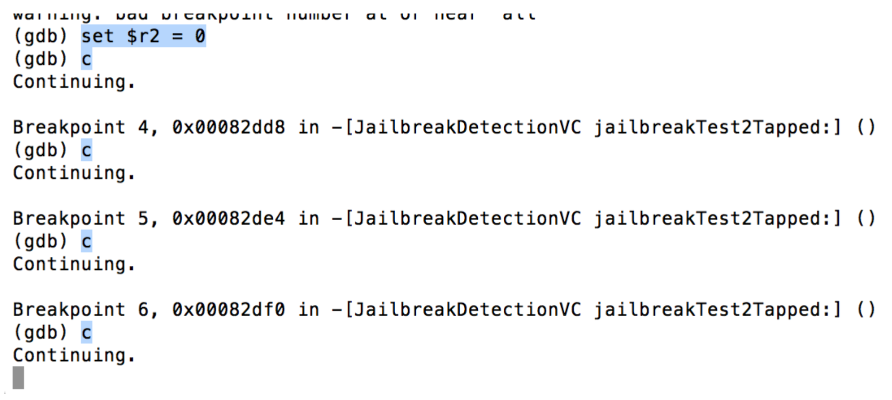
Well, there is something of interest almost immediately. We can see a method with the name *showAlertForJailbreakTestIsJailbroken:*. Let's not do anything with this and try to look up this method in the class information file. On looking up this method in the class information file, we can see the full prototype for this method.



```
@interface DamnVulnerableAppUtilities : /Users/Prateek/Desktop/DVIA/DamnVulnerableIOSApp/
DamnVulnerableIOSApp/View Controllers/
{
}

+ (void)addCommonBackgroundImageToViewController:(id)fp8;
+ (void)showAlertWithMessage:(id)fp8;
+ (void)showAlertForJailbreakTestIsJailbroken:(BOOL)fp8;
+ (void)pushWebVCWithURL:(id)fp8 viewController:(id)fp12;

@end
```

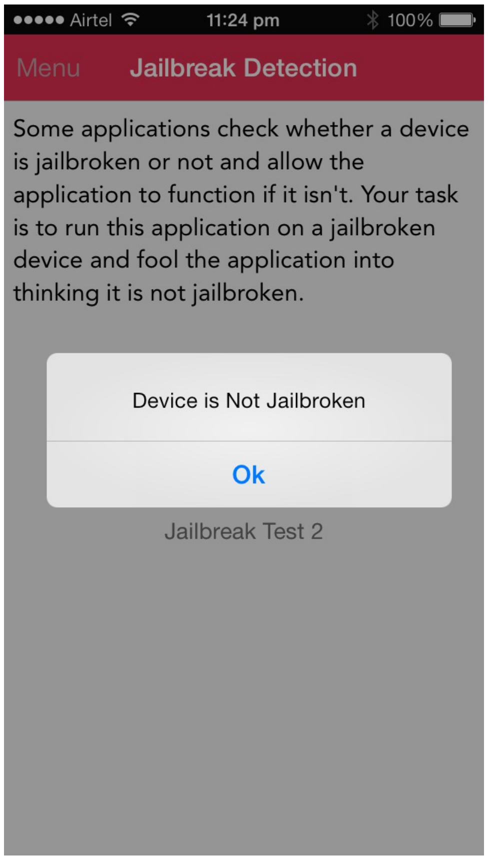
This method accepts a Boolean parameter. Well, this is what we have been looking for. All we need to do is pass the Bool parameter *NO* to this method. And if you are familiar with the basics of ARM and GDB, you will know that the first parameter goes inside the r2 register. If you don't understand this concept, I would recommend you read the article on ARM and GDB basics at http://highaltitudehacks.com/2013/11/08/ios-application-security-part-21-arm-and-gdb-basics/

So let's just set the value of r2 to 0 ( 1 = YES and 0 = NO) and type **c** to continue. Also, let's skip all the breakpoints after this by continuing.



```
(gdb) set $r2 = 0
(gdb) c
Continuing.

Breakpoint 4, 0x00082dd8 in -[JailbreakDetectionVC jailbreakTest2Tapped:] ()
(gdb) c
Continuing.

Breakpoint 5, 0x00082de4 in -[JailbreakDetectionVC jailbreakTest2Tapped:] ()
(gdb) c
Continuing.

Breakpoint 6, 0x00082df0 in -[JailbreakDetectionVC jailbreakTest2Tapped:] ()
(gdb) c
Continuing.
```

As you can see on the device, it says “Device is Not Jailbroken” even though our device is jailbroken. We just bypassed the check for a jailbroken device.



Some applications check whether a device is jailbroken or not and allow the application to function if it isn't. Your task is to run this application on a jailbroken device and fool the application into thinking it is not jailbroken.

Device is Not Jailbroken

Many developers check for a jailbroken device in their application and disable the application if the device is jailbroken. This is because critical applications are subject to risk if the device on which they run is jailbroken. However, no technique for detecting a jailbroken device is foolproof and can be bypassed by using such techniques.