Damn Vulnerable IOS Application Solutions

http://damnvulnerableiosapp.com/

Runtime Manipulation – Login Method 2

As you may have noted by now, the solution for *Login Method 1* doesn't work on *Login Method 2*. In order to figure out what must be going on, let's have a look at the class information for this application and check the methods for the view controller *RuntimeManipulationDetailsVC*

```
-
@interface RuntimeManipulationDetailsVC : /Users/Prateek/Desktop/DVIA/DamnVulnerableIOSApp/DamnVulnerableIOSApp/View Controllers/
{
    UITextField *_usernameTextField;
    UITextField *_passwordTextField;
}

- (void)setPasswordTextField:(id)fp8;
- (id)passwordTextField;
- (void)setUsernameTextField:(id)fp8;
- (id)usernameTextField;
- (void).cxx_destruct;
- (void)showLoginFailureAlert;
- (void)pushSuccessPage;
- (BOOL)isLoginValidated;
- (void)loginMethod3Tapped:(id)fp8;
- (void)loginMethod2Tapped:(id)fp8;
- (void)loginMethod1Tapped:(id)fp8;
- (void)didReceiveMemoryWarning;
- (void)viewDidLoad;
- (id)initWithNibName:(id)fp8 bundle:(id)fp12;

@end
```

We can safely assume that the method which gets called on tapping the button *Login Method 2* is

– (void)loginMethod2Tapped:(id)fp8

In order to understand what's happening in this method, we must analyze the application using GDB. So let's start GDB on our device . After starting GDB, make sure our application is running in foreground and attach to to it.

```
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.388
Attaching to process 388.
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/Xco
rableIOSApp.build/Debug-iphoneos/DamnVulnerableIOSApp.build/Objects-normal/armv7s/SolutionsViewController.o" -

warning: Could not find object file "/Users/Prateek/Library/Developer/Xcode/DerivedData/DamnVulnerableIOSApp-e
VulnerableIOSApp.build/Objects-normal/armv7s/RuntimeManipulationDetailsVC.o" - no debug information available

warning: Could not find object file "/Users/Prateek/Library/Developer/Xcode/DerivedData/DamnVulnerableIOSApp-e
VulnerableIOSApp.build/Objects-normal/armv7s/TransportLayerProtectionVC.o" - no debug information available fo

warning: Could not find object file "/Users/Prateek/Library/Developer/Xcode/DerivedData/DamnVulnerableIOSApp-e
VulnerableIOSApp.build/Objects-normal/armv7s/RNOpenSSLCryptor.o" - no debug information available for "RNOpenS
```

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 loginMethod2Tapped:*

```
(gdb) b loginMethod2Tapped:
[0] cancel
[1] all

Non-debugging symbols:
[2]    -[ApplicationPatchingDetailsVC loginMethod2Tapped:]
[3]    -[RuntimeManipulationDetailsVC loginMethod2Tapped:]
> 3
Breakpoint 1 at 0xd37ba
(gdb)
```

Now use 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 *Login Method 2.* We will see our breakpoint being hit.

```
(gdb) c
Continuing.

Breakpoint 1, 0x000d37ba in -[RuntimeManipulationDetailsVC loginMethod2Tapped:] ()
(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 (including background objc_msgSend calls). We should only be concerned with the *objc_msgSend* calls related to this method. So let's 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. As you can see, I am noting down the addresses of all the *blx* instructions ....

```
InList — ssh — 204:
ssh

0x000d37d0
0x000d37fe
0x000d3802
0x000d382a
0x000d382e
0x000d3866
0x000d38a2
0x000d38d8
0x000d38dc
0x000d3922
0x000d3942
0x000d3952
0x000d3958
0x000d395e
0x000d399c
0x000d39c0
0x000d39ca

0>:   ldr     r0, [sp, #8]
2>:   tst.w   r0, #1  ; 0x1
6>:   beq.n   0xd3974 <-[RuntimeManipulationI
8>:   movs    r0, #1
0>:   movt    r0, #0  ; 0x0
4>:   strb.w  r0, [sp, #68]
8>:   ldrsb.w r0, [sp, #68]
2>:   cmp     r0, #0
4>:   beq.n   0xd39a0 <-[RuntimeManipulationI
6>:   movw    r0, #38784      ; 0x9780
0>:   movt    r0, #2  ; 0x2
4>:   add     r0, pc
6>:   ldr     r0, [r0, #0]
8>:   movw    r1, #60104      ; 0xeac8
2>:   movt    r1, #2  ; 0x2
6>:   add     r1, pc
8>:   ldr     r2, [sp, #80]
0>:   ldr     r1, [r1, #0]
2>:   str     r0, [sp, #4]
4>:   mov     r0, r2
6>:   ldr     r2, [sp, #4]
8>:   blx     r2
0>:   b.n     0xd39c2 <-[RuntimeManipulationI
0x000d39a0 <-[RuntimeManipulationDetailsVC loginMethod2Tapped:]+492>:   movw    r0, #38748      ; 0x975c
0x000d39a4 <-[RuntimeManipulationDetailsVC loginMethod2Tapped:]+496>:   movt    r0, #2  ; 0x2
0x000d39a8 <-[RuntimeManipulationDetailsVC loginMethod2Tapped:]+500>:   add     r0, pc
0x000d39aa <-[RuntimeManipulationDetailsVC loginMethod2Tapped:]+502>:   ldr     r0, [r0, #0]
0x000d39ac <-[RuntimeManipulationDetailsVC loginMethod2Tapped:]+504>:   movw    r1, #60072      ; 0xeaa8
0x000d39b0 <-[RuntimeManipulationDetailsVC loginMethod2Tapped:]+508>:   movt    r1, #2  ; 0x2
0x000d39b4 <-[RuntimeManipulationDetailsVC loginMethod2Tapped:]+512>:   add     r1, pc
0x000d39b6 <-[RuntimeManipulationDetailsVC loginMethod2Tapped:]+514>:   ldr     r2, [sp, #80]
0x000d39b8 <-[RuntimeManipulationDetailsVC loginMethod2Tapped:]+516>:   ldr     r1, [r1, #0]
0x000d39ba <-[RuntimeManipulationDetailsVC loginMethod2Tapped:]+518>:   str     r0, [sp, #0]
0x000d39bc <-[RuntimeManipulationDetailsVC loginMethod2Tapped:]+520>:   mov     r0, r2
0x000d39be <-[RuntimeManipulationDetailsVC loginMethod2Tapped:]+522>:   ldr     r2, [sp, #0]
0x000d39c0 <-[RuntimeManipulationDetailsVC loginMethod2Tapped:]+524>:   blx     r2
0x000d39c2 <-[RuntimeManipulationDetailsVC loginMethod2Tapped:]+526>:   add     r0, sp, #72
0x000d39c4 <-[RuntimeManipulationDetailsVC loginMethod2Tapped:]+528>:   movs    r1, #0
0x000d39c6 <-[RuntimeManipulationDetailsVC loginMethod2Tapped:]+530>:   movt    r1, #0  ; 0x0
0x000d39ca <-[RuntimeManipulationDetailsVC loginMethod2Tapped:]+534>:   blx     0xfcf60 <dyld_stub_objc_storeS
0x000d39ce <-[RuntimeManipulationDetailsVC loginMethod2Tapped:]+538>:   add     sp, #84
0x000d39d0 <-[RuntimeManipulationDetailsVC loginMethod2Tapped:]+540>:   pop     {r4, r7, pc}
0x000d39d2 <-[RuntimeManipulationDetailsVC loginMethod2Tapped:]+542>:   nop
End of assembler dump.
(gdb) b *0x000d399c
Breakpoint 2 at 0xd399c
```

And now let's set a breakpoint on all of them.

```
(gdb) b *0x000d37d0
Breakpoint 5 at 0xd37d0
(gdb) b *0x000d37fe
Breakpoint 6 at 0xd37fe
(gdb) b *0x000d3802
Breakpoint 7 at 0xd3802
(gdb) b *0x000d382a
Breakpoint 8 at 0xd382a
(gdb) b *0x000d382e
Breakpoint 9 at 0xd382e
(gdb) b *0x000d3866
Breakpoint 10 at 0xd3866
(gdb) b *0x000d38a2
Breakpoint 11 at 0xd38a2
(gdb) b *0x000d38d8
Breakpoint 12 at 0xd38d8
(gdb) b *0x000d38dc
Breakpoint 13 at 0xd38dc
(gdb) b *0x000d3922
Breakpoint 14 at 0xd3922
(gdb) b *0x000d3942
Breakpoint 15 at 0xd3942
(gdb) b *0x000d3952
Breakpoint 16 at 0xd3952
(gdb) b *0x000d3958
Breakpoint 17 at 0xd3958
(gdb) b *0x000d395e
Breakpoint 18 at 0xd395e
(gdb) b *0x000d399c
Breakpoint 19 at 0xd399c
(gdb) b *0x000d39c0
Breakpoint 20 at 0xd39c0
(gdb) b *0x000d39ca
Breakpoint 21 at 0xd39ca
(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 values of registers and see if there is anything of interest. We are printing out the value of *r1* register with every *objc_msgSend* call here. If there is nothing of interest, we just type **c** to continue until the next breakpoint is hit.

```
Breakpoint 5, 0x000d37d0 in -[RuntimeManipulationDetailsVC loginMethod2Tapped:] ()
(gdb) x/s $r1
0x156daad0:      "\b\036?:`?l\025"
(gdb) c
Continuing.

Breakpoint 6, 0x000d37fe in -[RuntimeManipulationDetailsVC loginMethod2Tapped:] ()
(gdb) x/s $r1
0x338fdfcb:      "usernameTextField"
(gdb) c
Continuing.

Breakpoint 7, 0x000d3802 in -[RuntimeManipulationDetailsVC loginMethod2Tapped:] ()
(gdb) x/s $r1
0x3c2fe230:      "??/<??/<\017a"
(gdb) c
Continuing.

Breakpoint 8, 0x000d382a in -[RuntimeManipulationDetailsVC loginMethod2Tapped:] ()
(gdb) x/s $r1
0x32944673:      "text"
(gdb) c
Continuing.

Breakpoint 9, 0x000d382e in -[RuntimeManipulationDetailsVC loginMethod2Tapped:] ()
(gdb) x/s $r1
0x3293050d:      "autorelease"
(gdb) c
Continuing.

Breakpoint 10, 0x000d3866 in -[RuntimeManipulationDetailsVC loginMethod2Tapped:] ()
(gdb) x/s $r1
0x3293014d:      "isEqualToString:"
(gdb)
```

We can see something of interest after a few breakpoints. As you can see, a comparison is made with a particular string as we can see the method call to *isEqualToString:*

Our task is to find out what string is the value being compared to. It could be the username or password. 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 print out the value of r2 register by using the command *po $r2*.

```
(gdb) po $r2
0xcc0a8 does not appear to point to a valid object.
```

Oops, looks like there is some problem here ! Well, there has to be some other way of bypassing this login check. Let's look at the disassembly carefully again. What we know is that the value in the username and/or password text fields are compared with strings and if they are equal then the user is logged in. We can see this kind of disassembly twice in this code.

```
0x000ad490 <-[RuntimeManipulationDetailsVC loginMethod2Tapped:]+96>:    tst.w   r0, #255        ; 0xff
0x000ad494 <-[RuntimeManipulationDetailsVC loginMethod2Tapped:]+100>:   beq.n   0xad500 <-[RuntimeManipulationDetailsVC loginMethod2Tapped:]+208>
0x000ad496 <-[RuntimeManipulationDetailsVC loginMethod2Tapped:]+102>:   movw    r0, #57222      ; 0xdf86
```

```
0x000ad4ee <-[RuntimeManipulationDetailsVC loginMethod2Tapped:]+190>:   tst.w   r5, #255        ; 0xff
0x000ad4f2 <-[RuntimeManipulationDetailsVC loginMethod2Tapped:]+194>:   beq.n   0xad50c <-[RuntimeManipulationDetailsVC loginMethod2Tapped:]+220>
0x000ad4f4 <-[RuntimeManipulationDetailsVC loginMethod2Tapped:]+196>:   movw    r0, #57108      ; 0xdf14
```

Please note that the address of these instructions may be different in your case.

If you have a little bit of knowledge about ARM, you will note that the instruction *tst* stands for *test* whereas *beq* stands for *branch if equal.* Basically the test instruction performs a bitwise AND operation on the value in the first operand (r0 and r5 in our case) and the value of Operand2. The flow then branches to a particular address if the *branch if equal* returns a positive result, i.e if the values are equal.

Well, let's do a couple of things here.

a) Set a breakpoint before both the *beq* instructions.
b) Make sure the values in the register is set to 1 when these breakpoints are hit, this means setting r0 to 1 in case of first instruction and setting r5 to 1 in case of the 2nd instruction. This might help us bypass the login check as we are returning true with both the comparisons being made in this method.

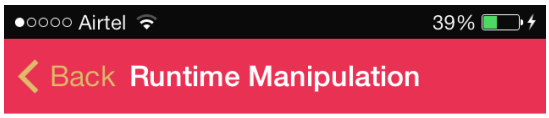
This is being demonstrated in the screenshot below.

```
(gdb) b *0x000bd490
Breakpoint 2 at 0xbd490
(gdb) b *0x000bd4ee
Breakpoint 3 at 0xbd4ee
(gdb) c
Continuing.

Breakpoint 2, 0x000bd490 in -[RuntimeManipulationDetailsVC loginMethod2Tapped:] ()
(gdb) info registers
r0             0x0      0
r1             0x40     64
r2             0x0      0
r3             0x1      1
r4             0x155969e0       358181344
r5             0x30a2314d       815935821
r6             0x155baf10       358330128
r7             0x27d4c53c       668255548
r8             0x388d52e0       948785888
r9             0x15b7b890       364361872
r10            0x15553fc0       357908416
r11            0x30a37673       816019059
r12            0x387a56ec       947541740
sp             0x27d4c524       668255524
lr             0x2dbf15ef       767497711
pc             0xbd490  775312
cpsr           0x80000030       2147483696
(gdb) set $r0 = 1
(gdb) c
Continuing.

Breakpoint 3, 0x000bd4ee in -[RuntimeManipulationDetailsVC loginMethod2Tapped:] ()
(gdb) set $r5 = 1
(gdb) c
Continuing.
```

And if we continue the app, we can see that we have successfully bypassed the authentication check.



Congratulations !! You have successfully bypassed the authentication check.