Payatu

Home › All Blogs › Sneha-Rajguru ›

IOS App Runtime Analysis Using GDB

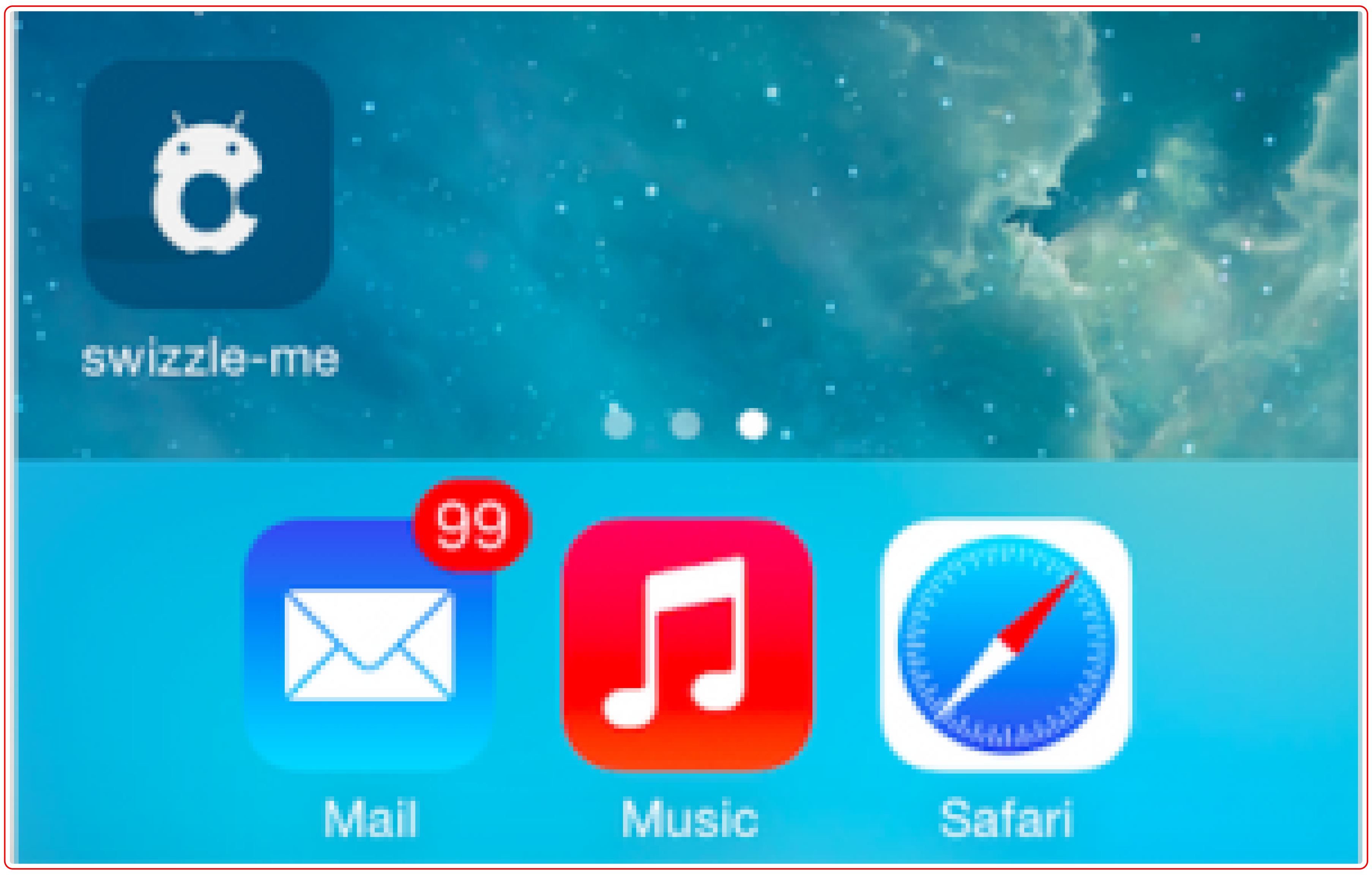
Sneha-Rajguru
16/06/2017

This blog is a simple guide for performing runtime analysis on iOS apps using GDB. With use of GDB we can get an in-depth knowledge of the application and not restricting to that, it also allows us to set breakpoints and manipulate the values and completely change the execution flow of the application.

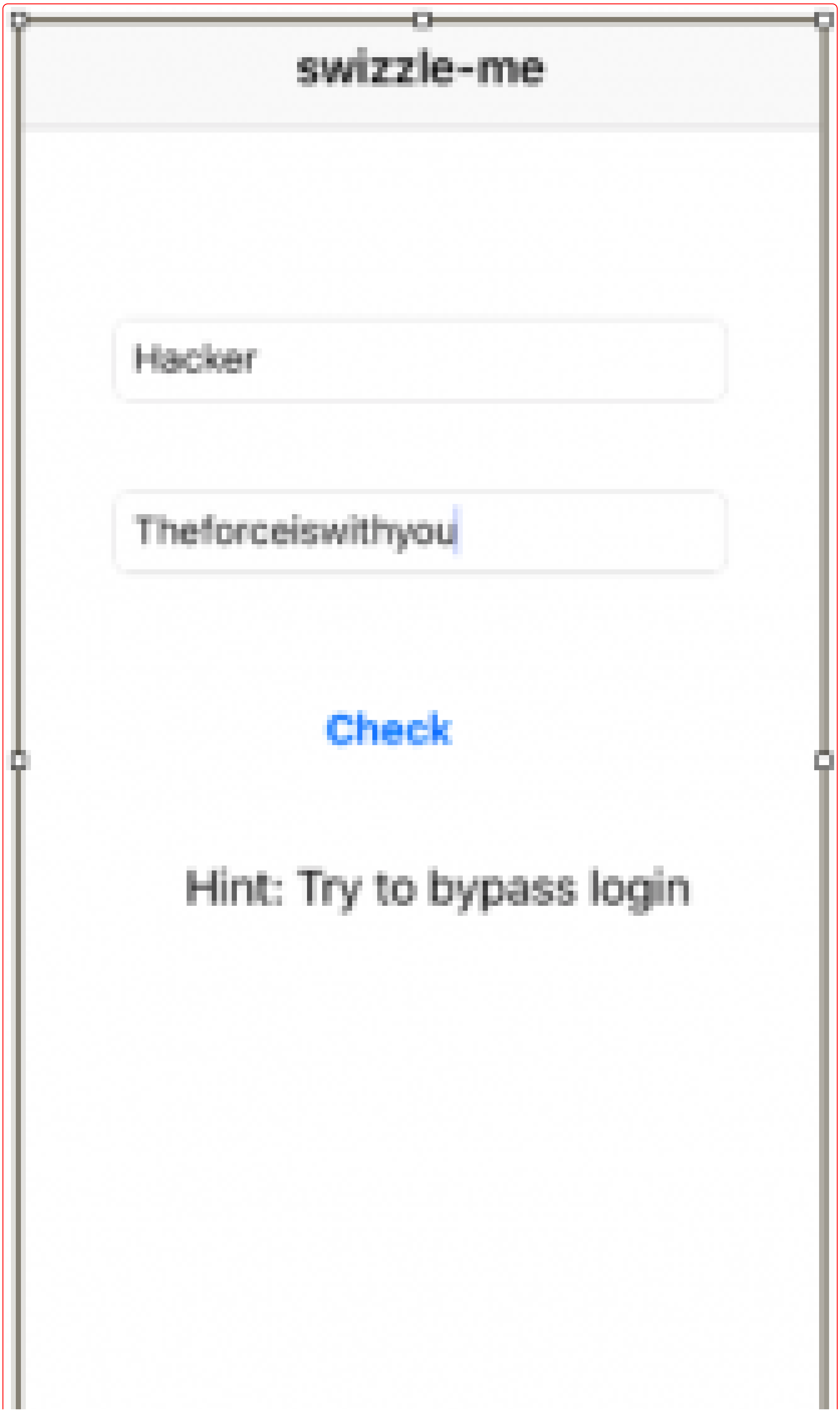
We have crafted a vulnerable iOS application for understanding and to learn to use GDB to perform runtime analysis, with our crafted vulnerable app known as 'swizzle-me'.

Introduction to the app.

The app 'swizzle-me' is a simple authentication app, wherein the user is required to enter his/her valid credentials and get access to the application.

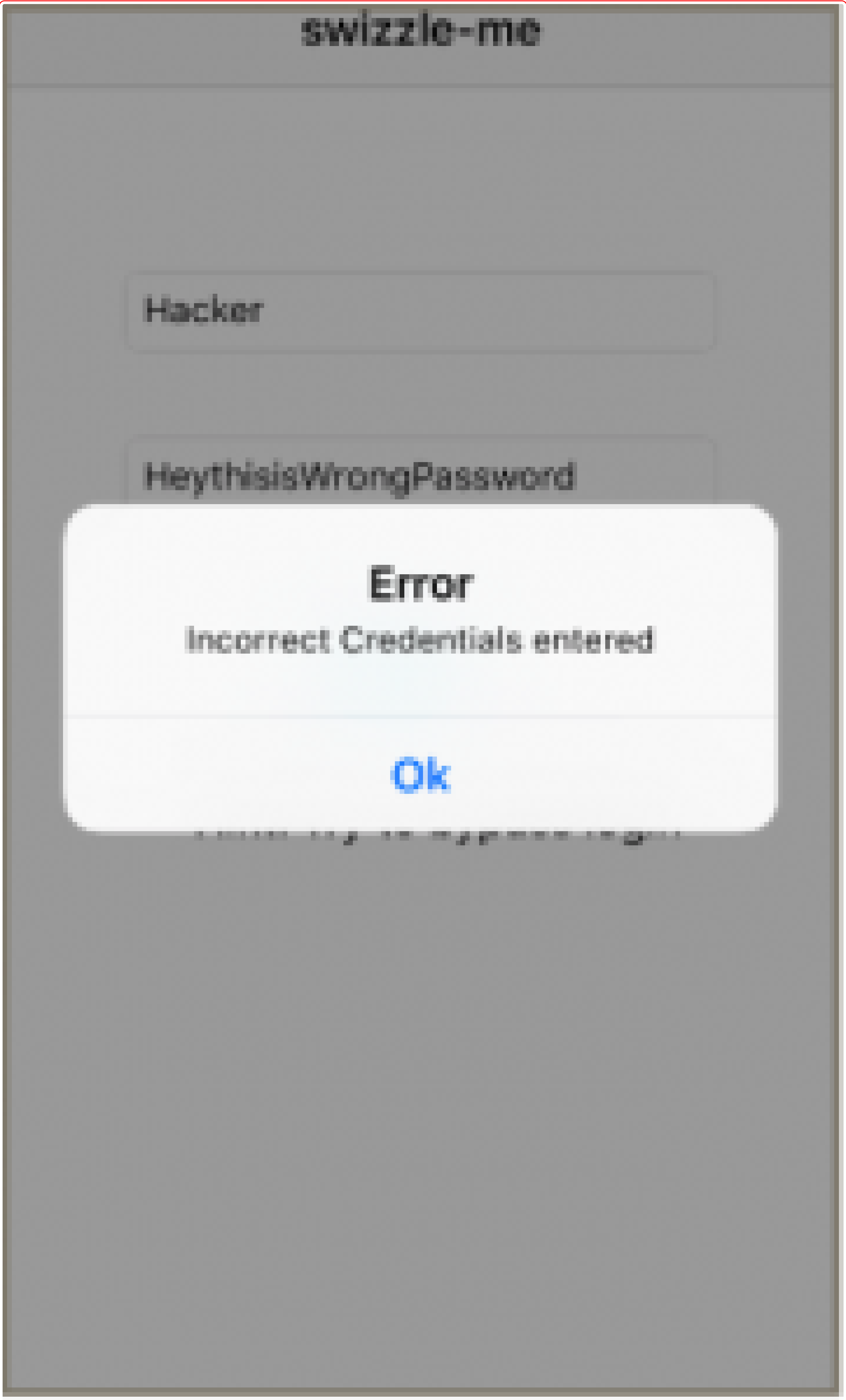


App's challenge: Your task is to bypass this login mechanism of the application and access the authenticated page!


swizzle-me
Hacker
Theforceiswithyou
Check
Hint: Try to bypass login

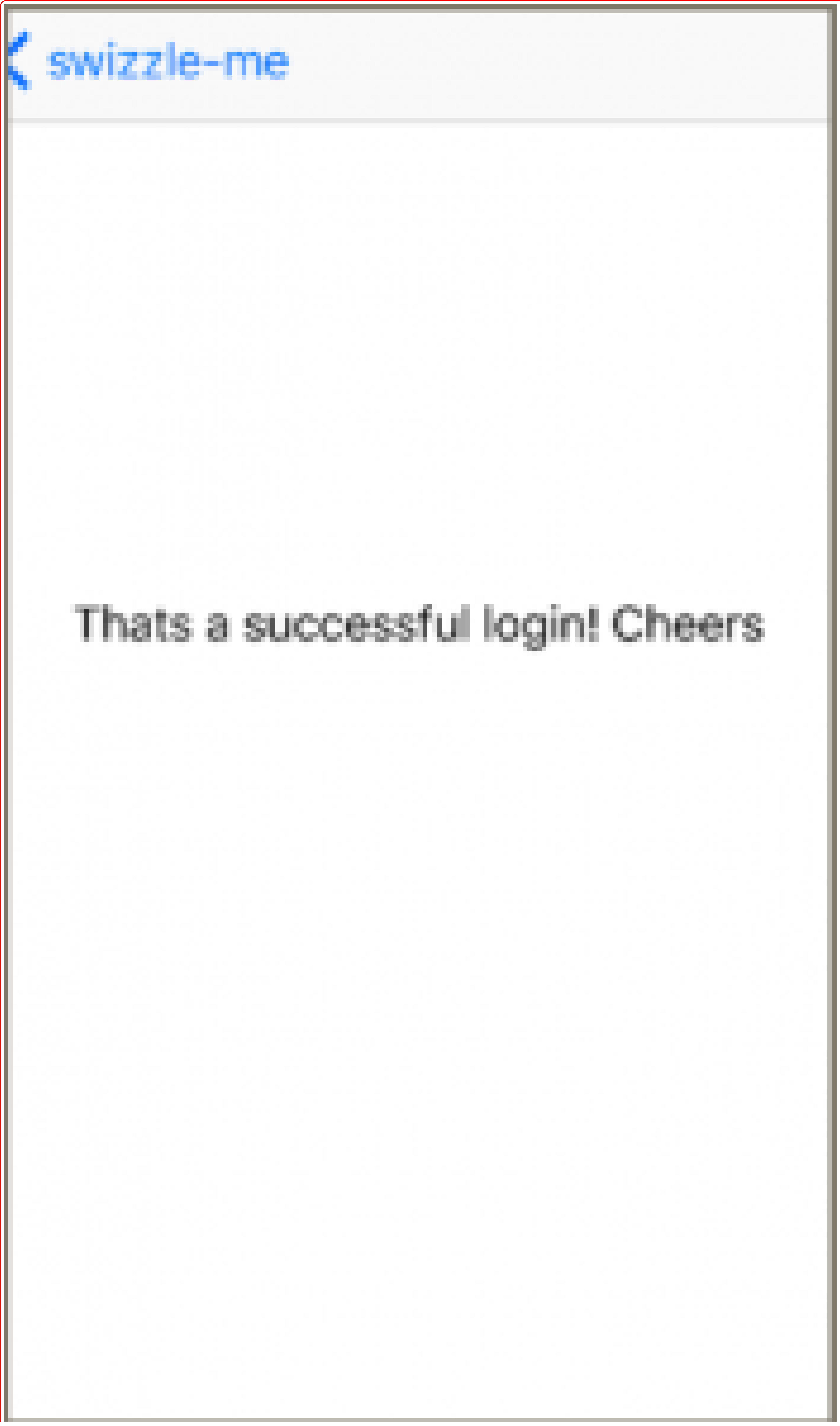
Screens of the application:

If wrong password is entered


swizzle-me
Hacker
HeythisisWrongPassword
Error
Incorrect Credentials entered
Ok

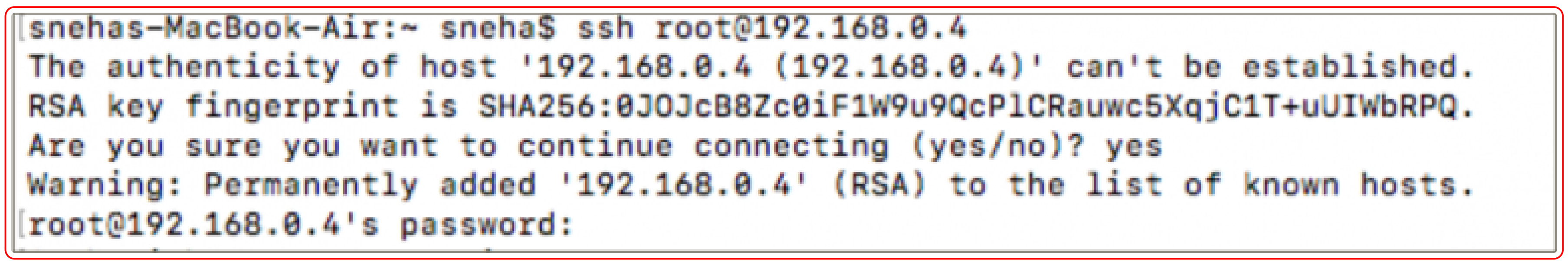
The application shows an error message stating the "Entered Credentials are incorrect".

Once, the correct credentials are entered the application greets the user with a 'successful login page'



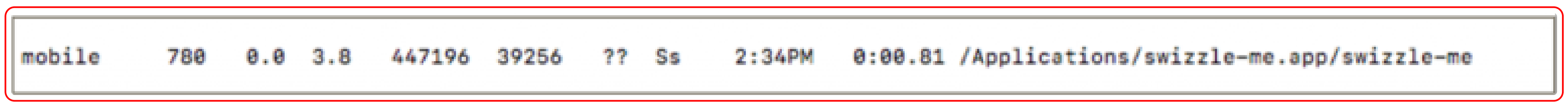
Thats a successful login! Cheers

Now login to the device:



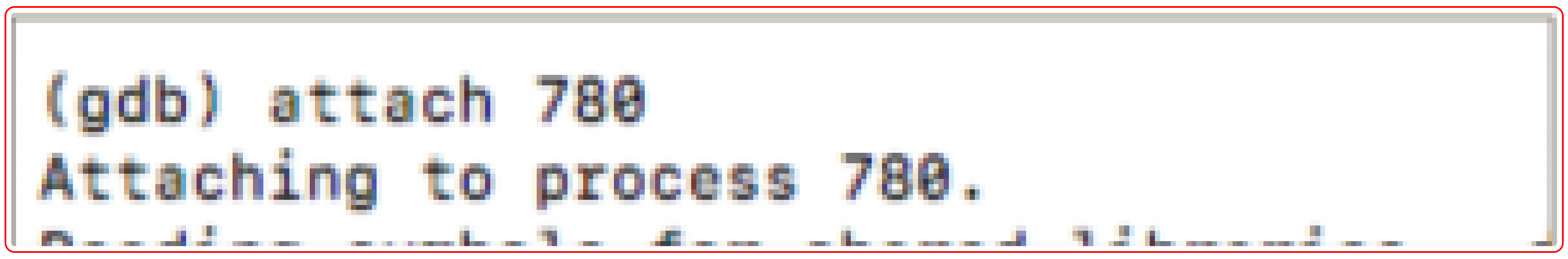
```
snehas-MacBook-Air:~ sneha$ ssh root@192.168.0.4
The authenticity of host '192.168.0.4 (192.168.0.4)' can't be established.
RSA key fingerprint is SHA256:0JOJcB8Zc0iF1W9u9QcPlCRauwc5XqjC1T+uUIWbRPQ.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.0.4' (RSA) to the list of known hosts.
root@192.168.0.4's password:
```

Now run the 'swizzle-me' app on your device. And attach GDB to the app's running process. For this use the command *attach gdb <pid_of_swizzle-me>,* here we got the pid for the app as *780*.



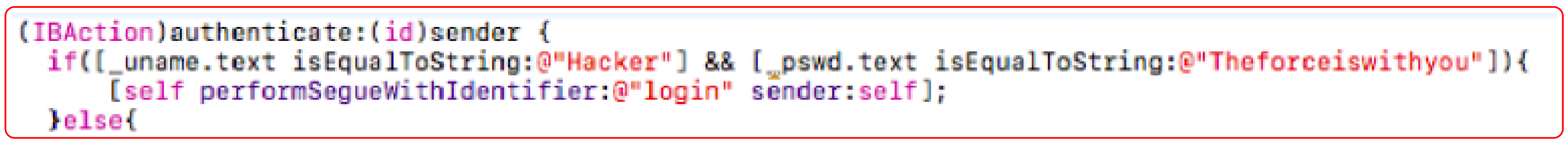
```
mobile      780   0.0  3.8   447196  39256   ??  Ss     2:34PM   0:00.81 /Applications/swizzle-me.app/swizzle-me
```

Now, lets attach the process.



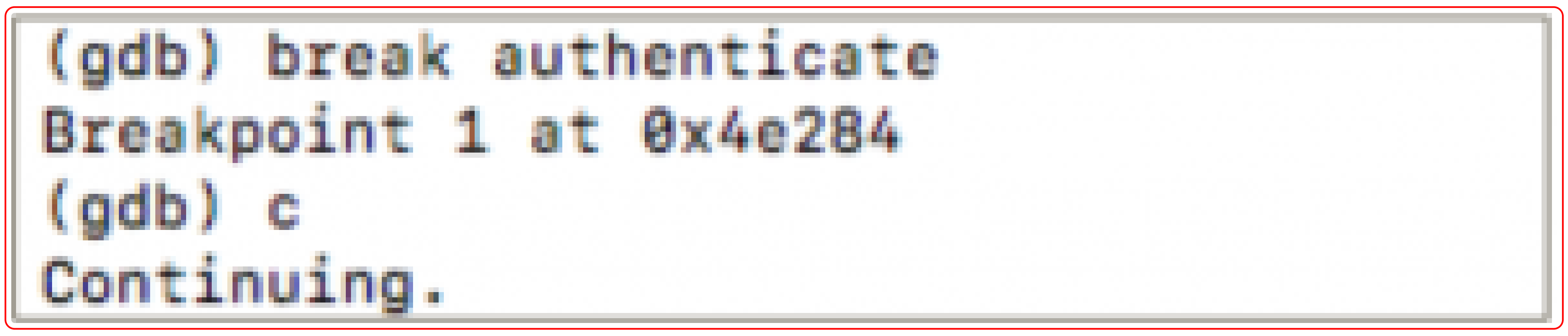
```
(gdb) attach 780
Attaching to process 780.
```

Before going ahead lets take a look at the code, and identify the method which is responsible for performing the authentication in the app



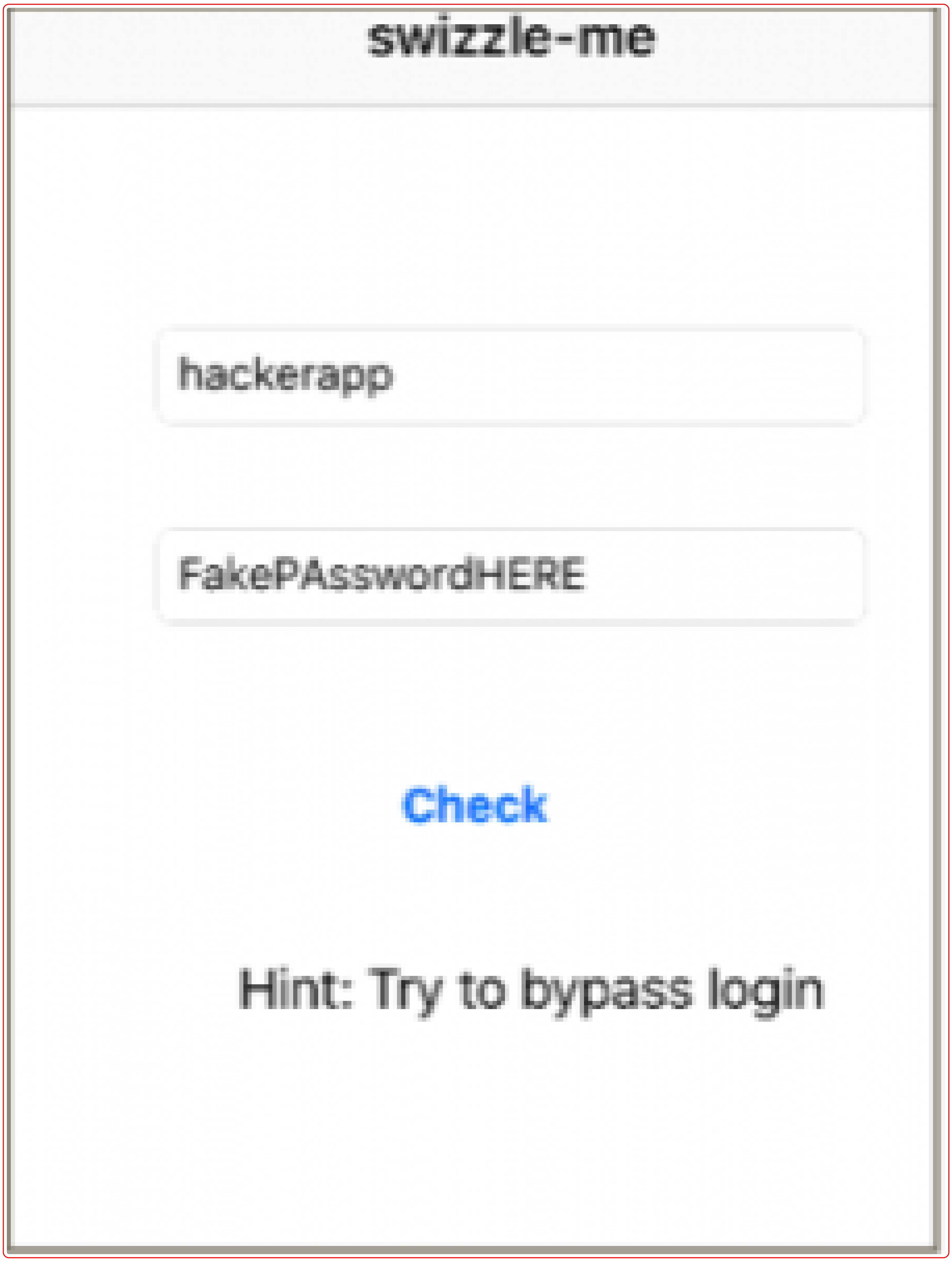
```
(IBAction)authenticate:(id)sender {
  if([_uname.text isEqualToString:@"Hacker"] && [_pswd.text isEqualToString:@"Theforceiswithyou"]){
      [self performSegueWithIdentifier:@"login" sender:self];
  }else{
```

You can use the class-dump-z to get the entire class dump of the application, there by gaining the knowledge of the methods of the application. Now, that we know the method responsible for authenticating the user, we shall put a **breakpoint** on this method. Method name "*authenticate*".



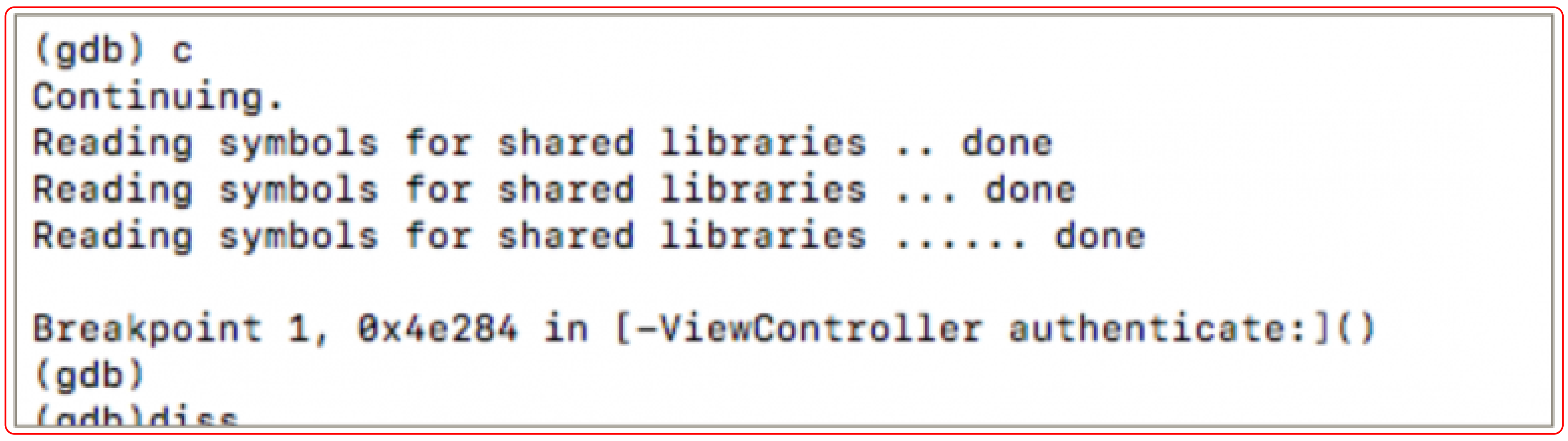
```
(gdb) break authenticate
Breakpoint 1 at 0x4e284
(gdb) c
Continuing.
```

As we have already set the breakpoint at the 'authenticate' method, lets enter any username and password in the app and press on check,



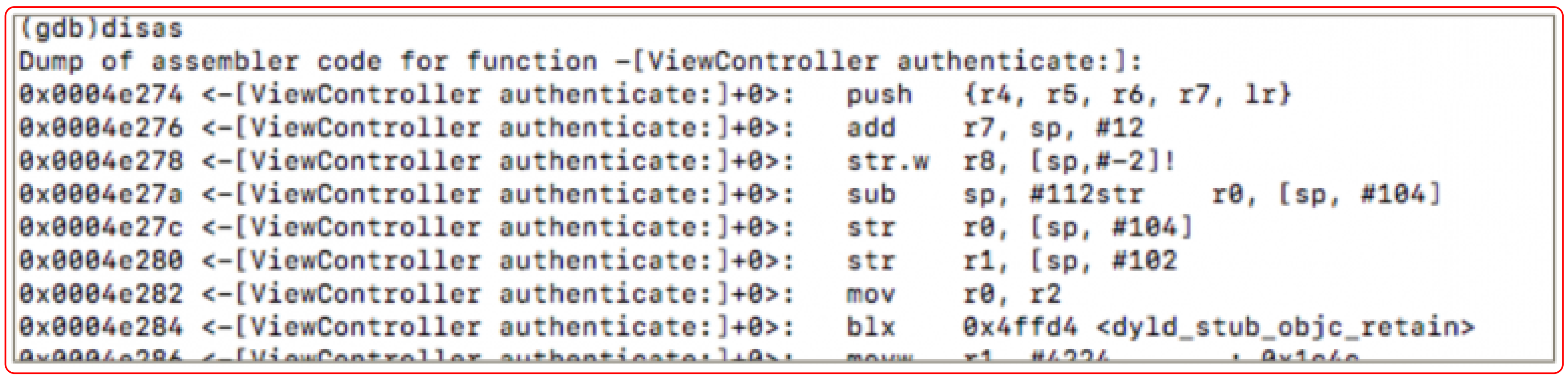
Now, use the *disas* to print the disassembly for this function.

And as it is known that the validation of the username and password is happening within this function (authenticate), and by looking at the disassembly we do not find any other interesting method related to our application.



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

Breakpoint 1, 0x4e284 in [-ViewController authenticate:]()
(gdb)
(gdb)diss
```

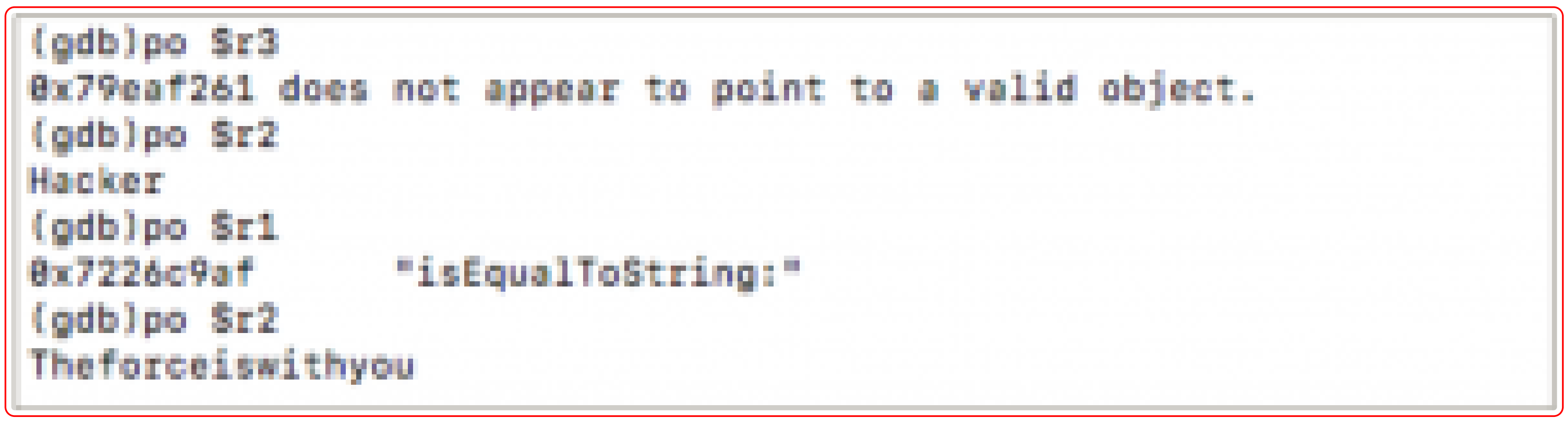


```
(gdb)disas
Dump of assembler code for function -[ViewController authenticate:]:
0x0004e274 <-[ViewController authenticate:]+0>:   push   {r4, r5, r6, r7, lr}
0x0004e276 <-[ViewController authenticate:]+0>:   add    r7, sp, #12
0x0004e278 <-[ViewController authenticate:]+0>:   str.w  r8, [sp,#-2]!
0x0004e27a <-[ViewController authenticate:]+0>:   sub    sp, #112str    r0, [sp, #104]
0x0004e27c <-[ViewController authenticate:]+0>:   str    r0, [sp, #104]
0x0004e280 <-[ViewController authenticate:]+0>:   str    r1, [sp, #102
0x0004e282 <-[ViewController authenticate:]+0>:   mov    r0, r2
0x0004e284 <-[ViewController authenticate:]+0>:   blx    0x4ffd4 <dyld_stub_objc_retain>
0x0004e286 <-[ViewController authenticate:]+0>:   movw   r1, #4224          ; 0x1a4e
```

Other way to look around for a method is to look for obj_msgSend function. Remember the obj_msgSend function is executed when an external function is called. Also, an app can have multiple obj_msgSend calls.

Considering our given scenario, we shall point out all the addresses of all those instructions who call the obj_msgSend, and put a breakpoint to it. A very simple way to do it is to look for the *blx* instruction, note its address and set a breakpoint for it and keep on pressing *c* (continue) until the next breakpoint is hit.

As we have set the breakpoint to our function, we now try to print out the values stored in it. Taking the advantage of objective-c, we understand that every object is a pointer.Thus providing pointer we will try to get inside the registers and see what value it has. To find out the value, we use '*po*' command to *print* the value of the object.

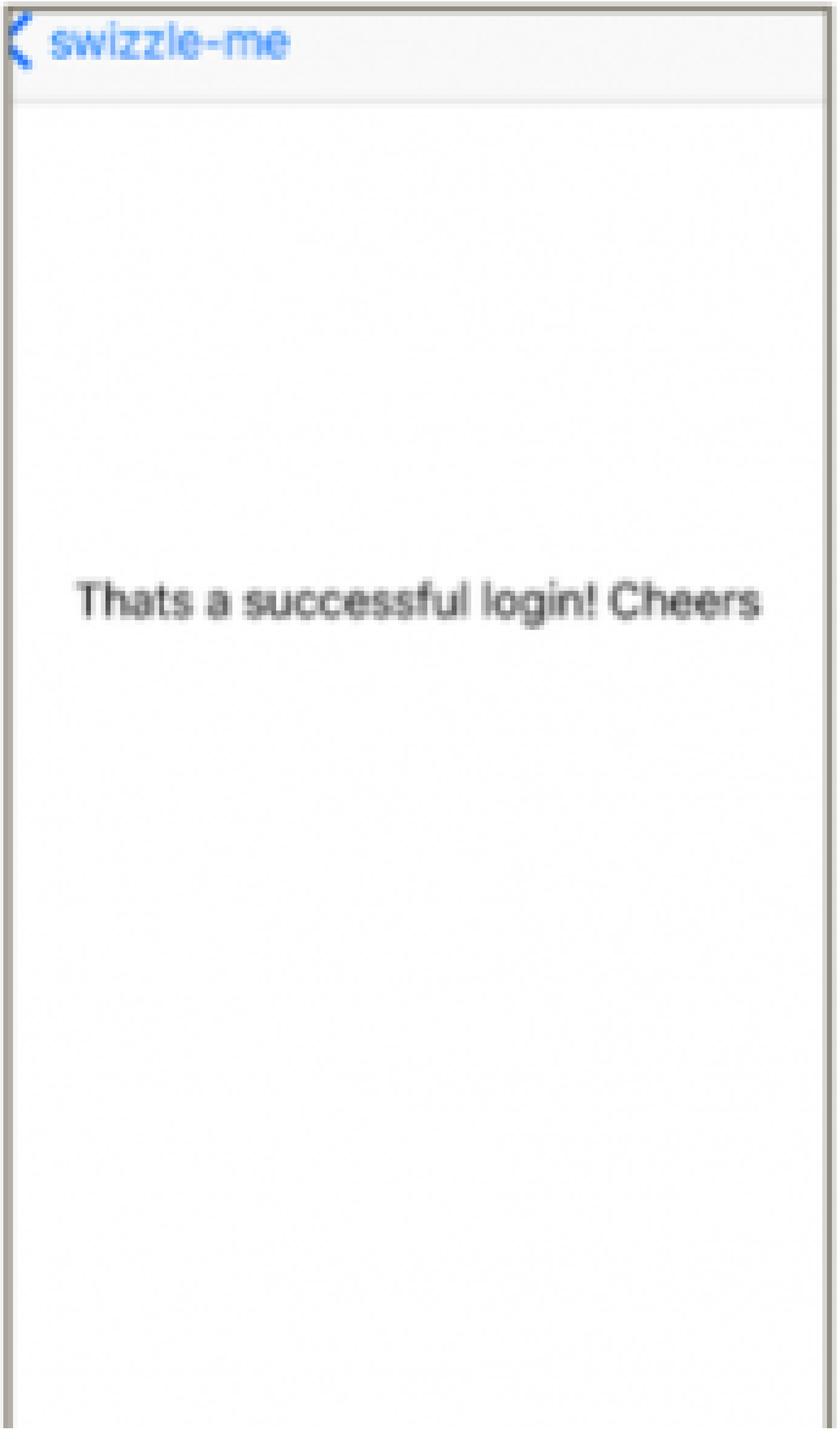


```
(gdb)po $r3
0x79eaf261 does not appear to point to a valid object.
(gdb)po $r2
Hacker
(gdb)po $r1
0x7226c9af       "isEqualToString:"
(gdb)po $r2
Theforceiswithyou
```

With the use of '*po*' command we could get the actual values of the objects.

As seen in the above image we have got the values 'Hacker' and 'Theforceiswithyou' , by looking at it we can surely tell that this must be the hardcoded username and password for the 'swizzle-me' app.

Our next step would be to try to enter the received values as app's credentials.



Thats a successful login! Cheers

Reference

https://blog.netspi.com/ios-tutorial-dumping-the-application-heap-from-memory/
http://resources.infosecinstitute.com/ios-application-security-part-22-runtime-analysis-manipulation-using-gdb/#gref
http://www.iicybersecurity.com/pentesting-cracking-analysis-ios-apps.html

Payatu

Get to know more about our process, methodology & team!

GET STARTED TODAY

All Blogs › Latest Blogs

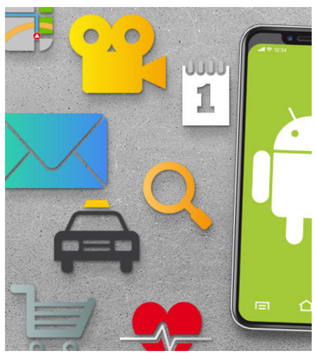
01/03/2021
Manmeet

Graphql Exploitation - Part 1- Understanding Graphql & Enur

24/02/2021
Surendra

Top 5 Cyber Attacks Of 2020

23/02/2021
Amit

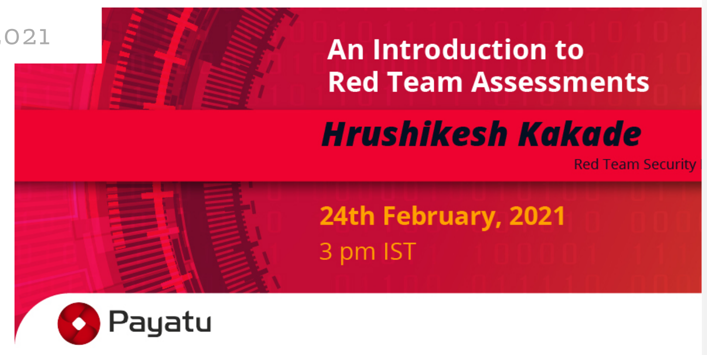


Android Security Part 1- Understanding Android Basics

☷ All News › Latest News

Webinar, Online

24-February-2021



Hrushikesh Kakade will be giving a talk on “An Introductio

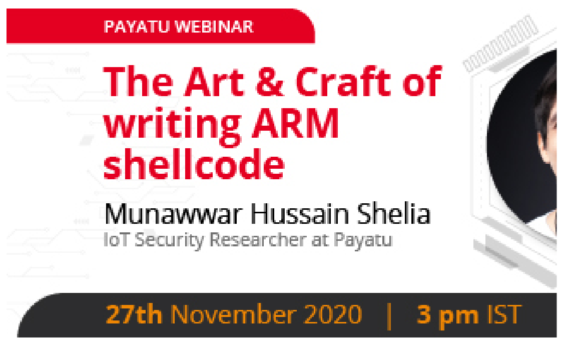
Webinar, Online

15-January-2021

Appar Thusoo will be giving a talk on “Under The Air:

Webinar, Online

27-November-2020


PAYATU WEBINAR
The Art & Craft of writing ARM shellcode
Munawwar Hussain Shelia
IoT Security Researcher at Payatu
27th November 2020 | 3 pm IST

Munawwar Hussain Shelia will be giving a talk on “The Art

Subscribe to Our Newsletter

Your E-Mail Address SUBSCRIBE

or

Follow our Social Media Handles

Follow @payatulabs 2,699 followers **Mi piace** 738

ABOUT US
ADVISORY
CAREER
BLOG
LATEST NEWS
DISCLOSURE-POLICY