**P632 - Spring 2016**

**InsecurePay**

**Cigital Project**

**Vulnerability Documentation**

Amish Shah (shah7)

Ankit Sadana (asadana)

Jaini Vora (voraj)

Janak Bhalla (jbhalla)

Madhukiran Radhakrishnan (madhradh)

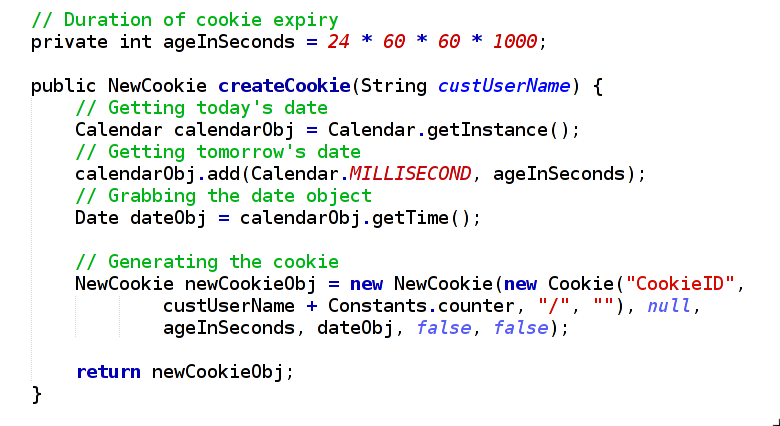
## **TABLE OF CONTENTS**

1. Generic 2
2. Login Module 5
3. Account Management Module 9
4. Transfer Funds Module 13
5. Activity History Module 15
6. Support Chat Module 17

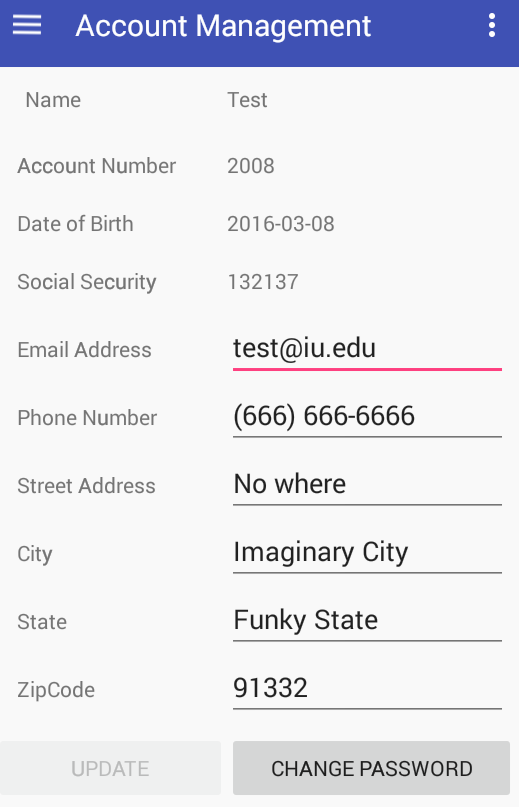
## 

## **Generic**

1. **Cookie - Secure(M9):** Secure flag in the cookie is set to false.
2. **Cookie - Expiry(M9):** Cookie expires after a day
3. **Cookie - HttpOnly(M9)**: HttpOnly attribute set to false

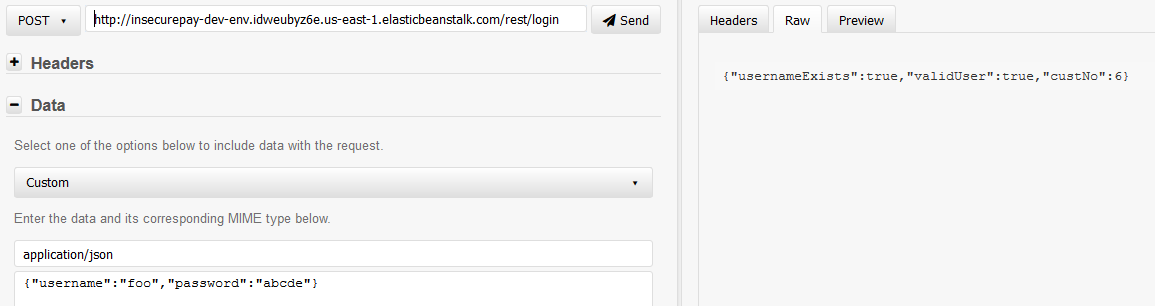


1. **Cookie - Privilege Escalation (Horizontal) (M5):** Cookie can be reset to any customer number through accountService and then can be used in Account Management.
2. **Cookie - Improper cookie handling(M9):** Cookies remain active after user logs out on the server side.
3. **Session Hijacking(M9):** Cookie ID can be predicted since it is set by using a counter and username
4. **Session Fixation(M9):** To produce a session fixation attack produce this vulnerability:
   1. From device1(attacker), Login as: username: *testUser*, Password: *12345*. And go to account management to check your details *(not required)*.

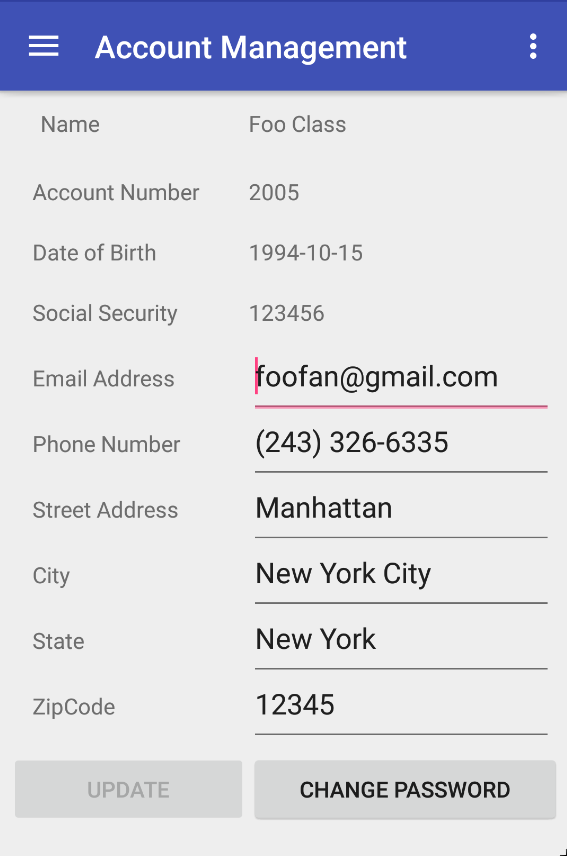


* 1. Let the device be idle for certain amount of time, by default 15 minutes.
  2. After 15 minutes, from device2(victim) login as: username: *foo*, Password: *abcde*

We used a REST debugger to login. Screenshot below:



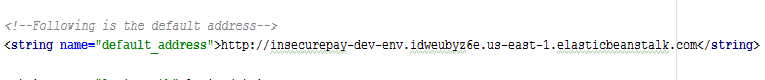
* 1. On device1, when the attacker goes back to Account Management page. The account details of ‘foo’ are displayed instead ‘testUser’



1. **Sensitive information sent as a query string parameter(M3):** Passing sensitive information such as Account Number in the URL. This can be detected by intercepting a request.
2. **Logout out check on back button pressed(Business Logic):** We are not logging out if the user goes out of the application by pressing the back button.
3. **Screenshots not disabled(M4):** Screenshots are enabled.

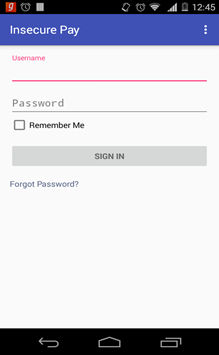
Screenshot_20160409_161343.png

1. **Http Connection(M3):**All requests are sent with the following address with http connection.

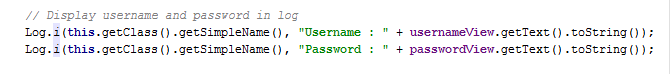


## **Login Module**

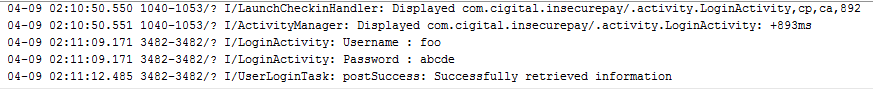
1. **Last Login not displayed(M9):** On the login page we are not displaying the time when user was last logged in.



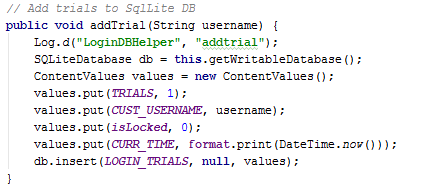
1. **Login credentials in logs(M4):** Once the user clicks the ‘Sign In’ button, the username and password get displayed in the logs. The following code displays the username and the password in the logs in LoginActivity.java



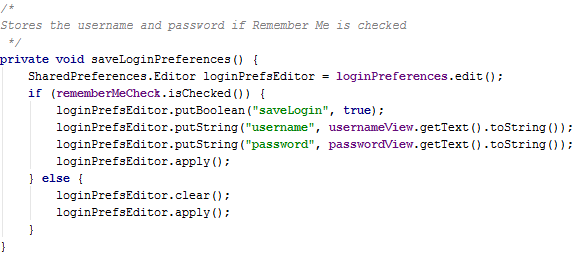
Username and password in the logs:



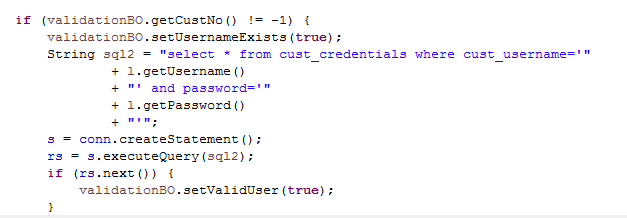
1. **Account Lockout(M2):** A SQLite Database in the user’s device stores the number of login attempts. The account gets locked after 15 attempts for 1 minute. The following code in the LoginDBHelper.java:



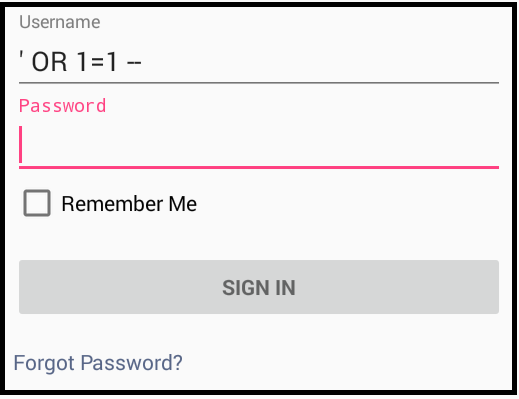
1. **Remember Me(M2):** SharedPreferences are used to store credentials in the user’s local storage if ‘Remember Me’ option is selected at login. By reviewing the source code in LoginActivity.java, it can be found that username and password are stored in shared preferences.



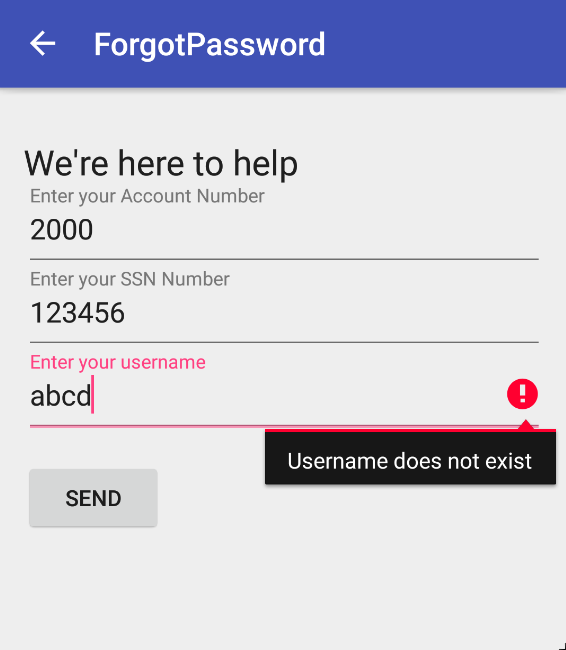
1. **SQL Injection(M7):** The query used to retrieve username and password is not parametrized to promote SQL injection. Following is code to retrieve username and password in ‘com.cigital.insecurepay.dao’ package > LoginDao.java



One can also try injecting sql injection string  **‘OR 1=1 - -** in the username field as shown in the image below.



1. **Login on Back Button press(M12):** To detect this vulnerability, sign in to the app, press back button twice and he/she can get into the app again.
2. **Username Enumeration(M5):** If an attacker tries to brute-force the username field then the attacker can have a list of all valid usernames. This can be detected by noticing that we are not limiting the number of attempts anywhere in ‘activity/ForgotPassword.java’ to enter username.



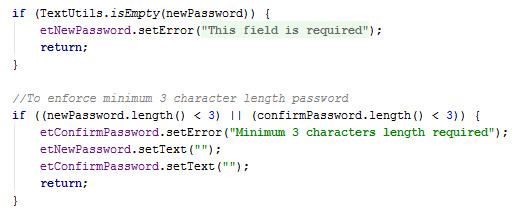
## 

## 

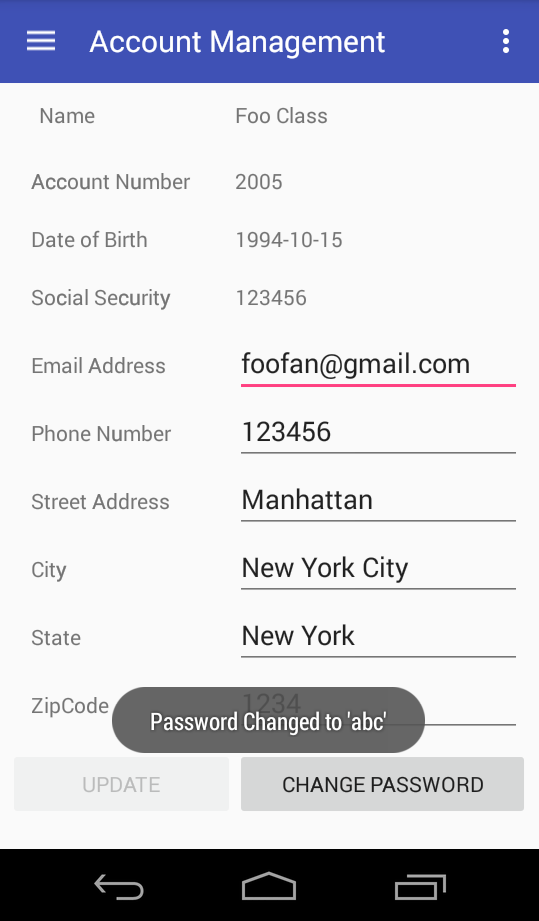
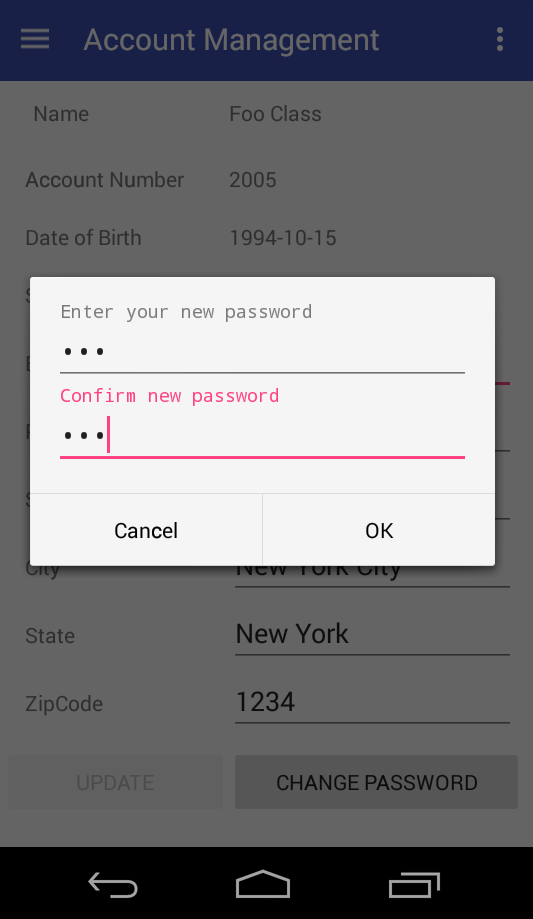
## **Account Management Module**

1. **Weak Password Policy(M5):** It can be detected in two ways:

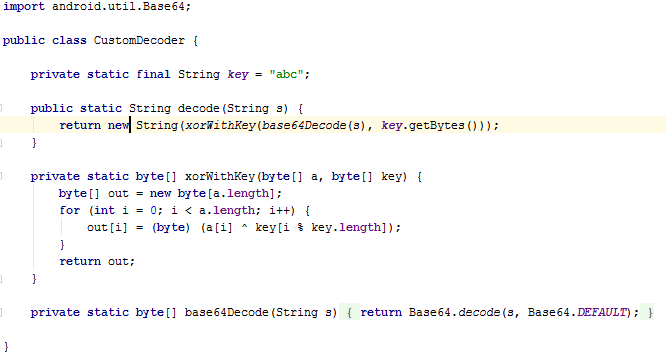
(1) By reviewing the code of AccountFragment.java



(2) While setting new password, enter three characters only and tap OK. The app will allow to reset the password.



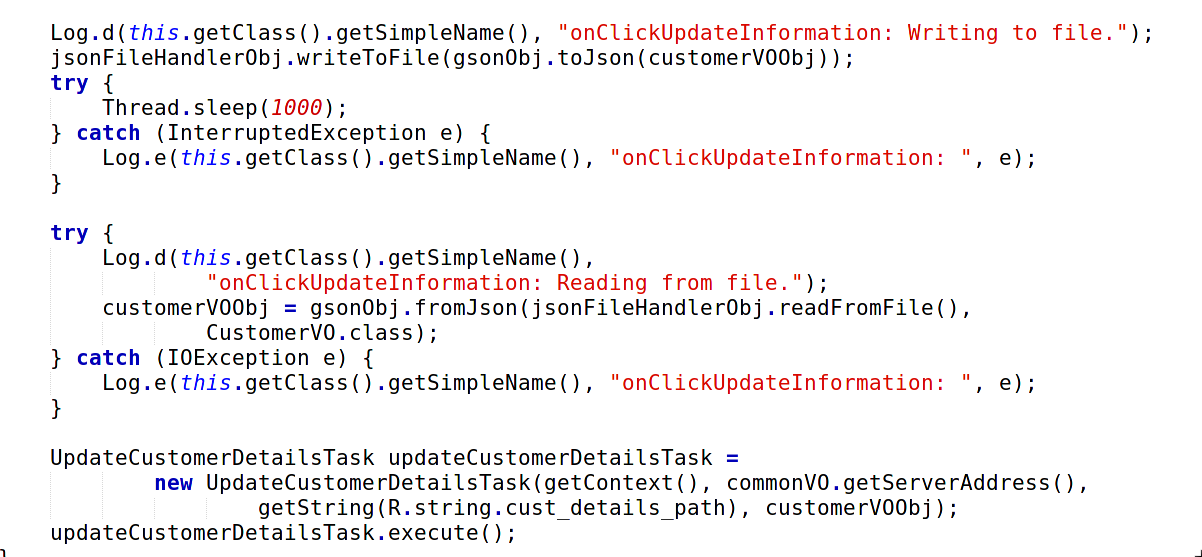
1. **Weak Cryptography(M6):** Code for decoding and decrypting SSN by XOR-ing with a key is present in CustomDecoder.java inside ‘common’ folder.



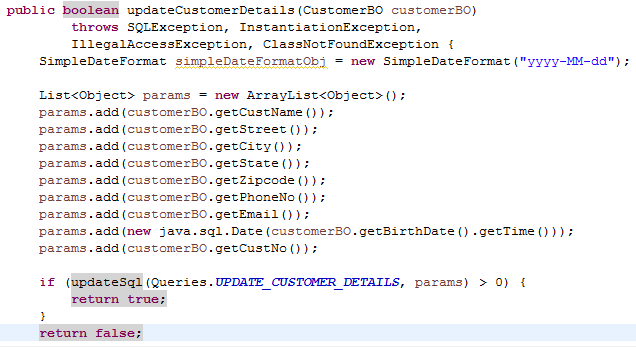
Code for encoding SSN and encrypting it by XOR-ing with key is present in a similar way at server-side. It is present in ‘com.cigital.insecurepay.common’ package > CustomEncoder.java

1. **Encryption/decryption key hard-coded(M2):** The key to decrypt SSN is present in CustomDecoder.java and can be seen in the image above. In a similar manner, a key to encrypt SSN is stored at the server side in ‘com.cigital.insecurepay.common’ package > CustomEncoder.java
2. **Horizontal Privilege Escalation(M5):** While updating the values of any customer from Account Fragment, all the information is read from a local JSON File. To perform this attack, the attacker can follow the following steps:
   1. Login using any user
   2. Access the account fragment
   3. Change any of the values
   4. The attacker can pause the app after clicking update and edit *custNo* in the JSON file created from the updated values.
   5. The JSON file can be found at:   
      *data>data>com.cigital.insecurepay>files>'username.json'*
   6. Doing this and unpausing the application will send the altered *custNo* to the server, and since there is no validation, the altered *custNo* will be updated with the JSON values instead of the current user.

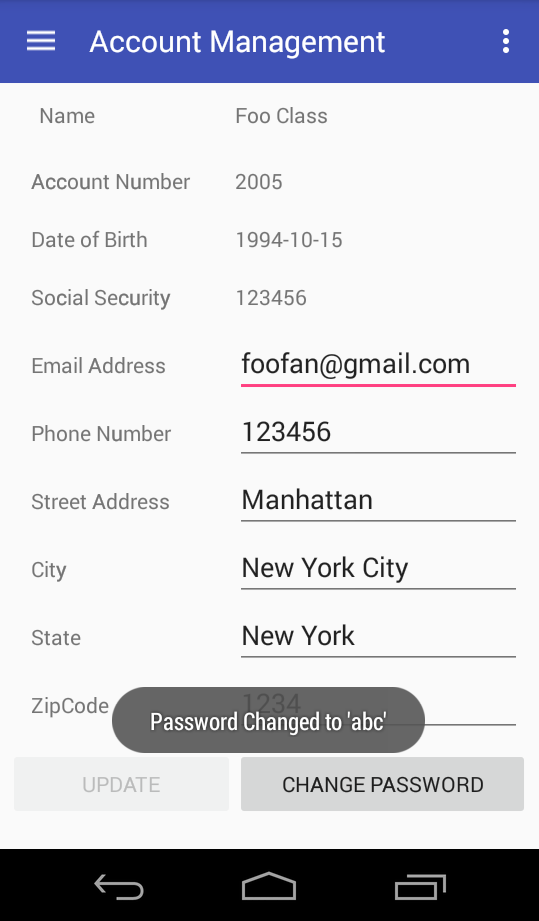
The following code is executed when a user clicks Update button in AccountFragment:



1. **Improper input validation(M1):** No server-side validation is performed on values that are being received from Account Management Activity. This can be observed in the following code in ‘com.cigital.insecurepay.dao’ package > CustomerDao.java

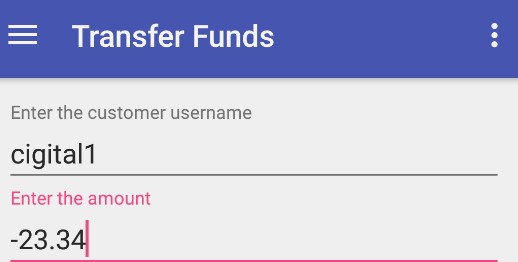


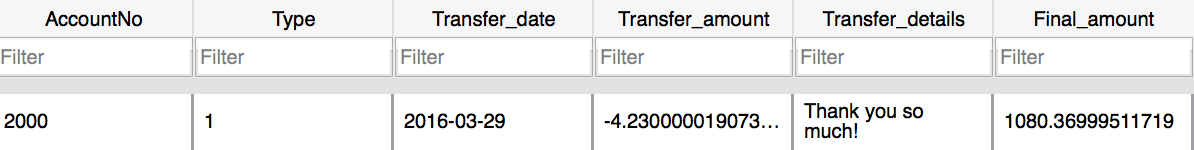
1. **Cleartext Password Response(M1):** Once a new password is set, server sends a response that includes a password in cleartext and a toast is displayed to the user which shows the new password.



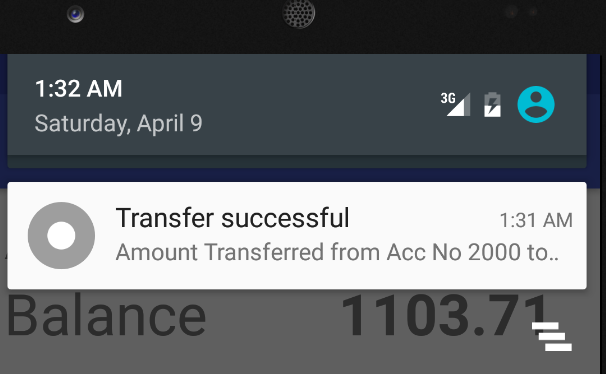
**Transfer Funds Module**

1. **Negative Funds Transfer(Business Logic):** No check is performed on amount field for negatives numbers.Negative amount for a transfer will credit amount to sender and debit from the receiver.

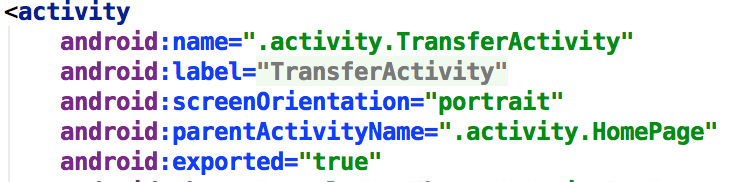




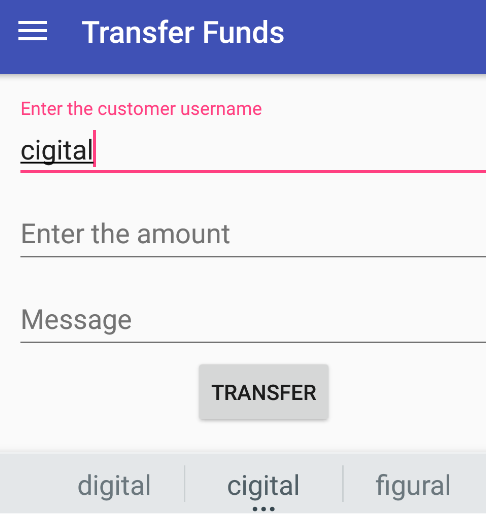
1. **Notification shows internal sensitive information(M4):** Notification shows sensitive sender and receiver account number.

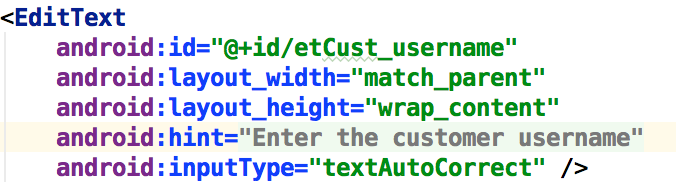


1. **Cross App Request Forgery(M8)**: Created a separate Transfer activity for confirming transfer whose ‘exported’ flag is set to true in AndroidManifest.xml.



1. **Keyboard Caching On (Device specific):** If username added to dictionary of the keyboard, then it will appear in its suggestions box while typing the username. For the customer username field, input type is made text autocorrect in fragment\_transfer.xml.



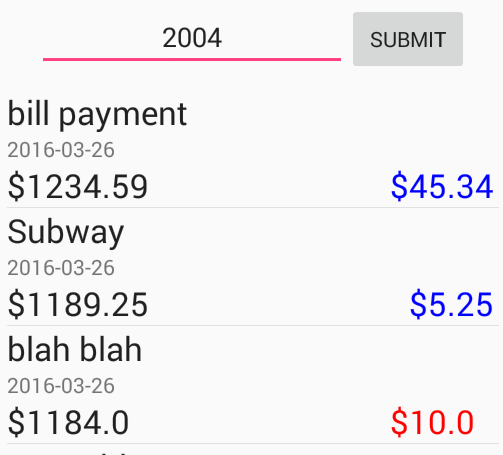
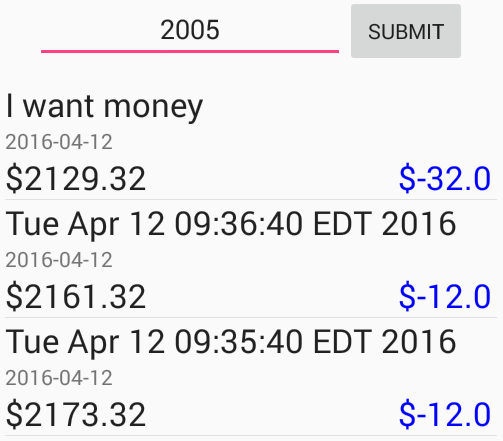


## 

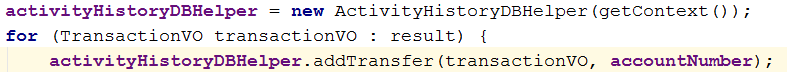
## 

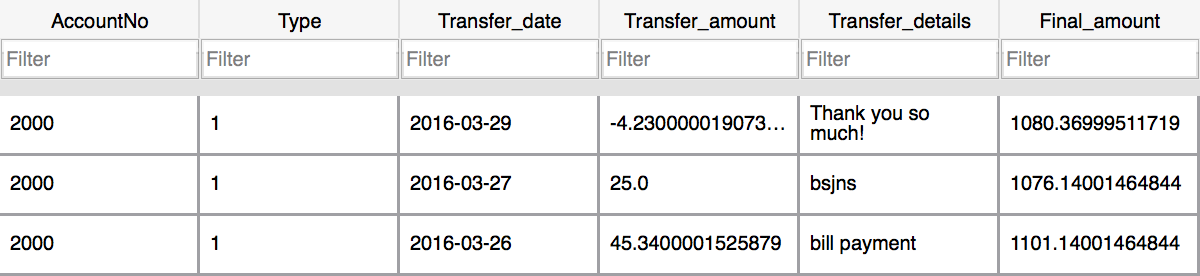
## **Activity History Module**

1. **Horizontal Privilege Escalation(M5)**: You can enter any valid account number in the text field and check that account’s activity history. No check is done on the server side that the account no of the service requestor matches with the account no entered in the text field. Following images shows activity history for two different account numbers from the same user.

1. **Dump sensitive information in local(M2):** Activity History gets dumped into local database using Activity History DB Helper. On post success of Activity History fetch task, all the transaction history is stored in database table ‘Transfers’.



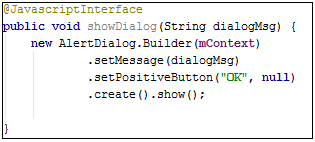
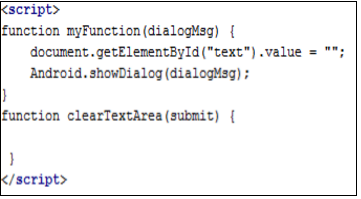


## 

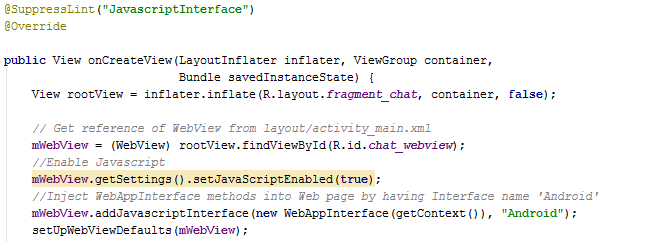
## 

## **Support Chat Module**

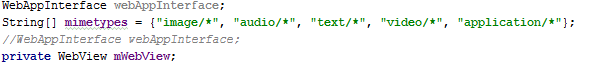
31. **JS Bridge:** LiveChat.html in the assets folder has a script that calls ChatFragment’s showDialog() function. This is possible because JSEnabled is set to true.



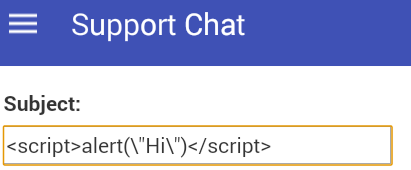
LiveChat.html ChatFragment.java



32. **Malicious File Upload:** Applications with all file types can be uploaded. This can be detected by observing the following code in ChatFragment.java or by uploading various types of files and if a toast for ‘File Upload Successful’ is shown then that particular file is uploaded without any restrictions.



33. **Cross-Site Scripting Chat:** Follow the steps below to conduct cross site scripting. After signing in, go to Chat. Enter some javascript in the subject:



Press the ‘Send!’ button. If the script is entered as displayed in the above image. Since the subject is sent back from the server as it is, the app will render it and display a pop-up:

