VulnApp – A vulnerable android application for learning

Aadhavan Anbuchezhian

May 2018

Submitted in part fulfilment for the degree of
B.Sc. Computing in Digital Forensics and Cyber Security
School of Informatics and Engineering,
Institute of Technology Blanchardstown,
Dublin, Ireland

Declaration

Acknowledgement

Author of this Project, Aadhavan Anbuchezhian would like to thank Christina Thorpe (Project supervisor), Mark Lane and Michael Hegarty for their continuous support and guidance through the duration of the year.

Table of Contents

Abstract

This report covers the development of a vulnerable android based application used for learning purposes. The main purpose of this application is to create awareness of android security threats among android developers and security enthusiasts. It achieves this goal by providing a learning portal where users can engage in bug hunting experience that improves their knowledge on exploits and their prevention and communicate together through discussions.

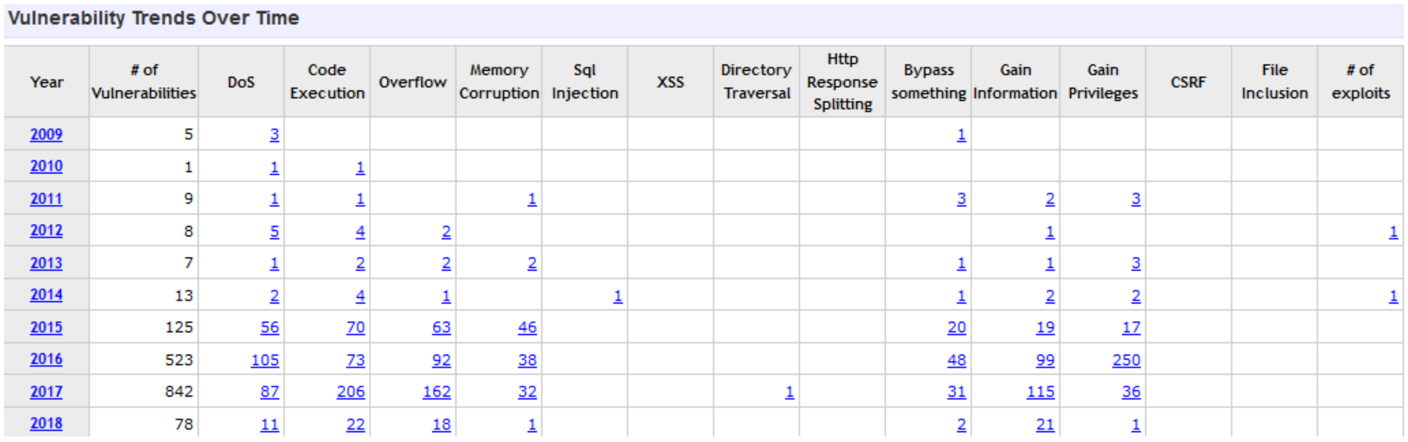
1 Introduction

The project application focuses on delivering a mobile application that offers a learning experience on the most common android vulnerabilities. The Mobile application bases its functionalities on introducing and education users on exploits that can arise along with an application development. The project is primarily focused on Android operating system and vulnerabilities within the OS. The program will feature several exploits pertaining to mobile application design and development. The knowledge of the examples of vulnerabilities shown in the program are targeted towards application developers who are unaware or interested in learning about issues that can be resolved by focusing on proper coding methods to reduce and combat common attacks.

2 Literature Review

2.1 General Research

Android operating system from the beginning has hosted many famous vulnerabilities such as Heartbleed (lookout, 2014) which affected the android devices by exploiting the Secure Socket Layer (SSL) communication. Heartbleed bug was used by attackers to steal information (64kb at once) from the device memory. This type of exploit is classified under the “Insecure Communication” category according to OWASP’s Top 10 mobile vulnerabilities (OWASP, 2016). Another infamous threats to appear on the android ecosystem is StageFright (Norton, 2015). This bug was named the most dangerous in android’s history uses Multimedia Messaging Service (MMS) as a vector for malicious code which is categorized as Remote Code Execution (RCE). Once the code is executed, the attack allows for the accessing of permission, data from storage etc. Risk types such as Code Execution are the most popular at 21% of all exploits according to Vulnerability database (CVE, 2018). Though android developers were able to fix the risk areas relatively soon, newer and obscure bugs are still being discovered on the Android Operating system (Fig.1 - CVE, 2018).



Vulnerability Trends Over Time

Year	# of Vulnerabilities	DoS	Code Execution	Overflow	Memory Corruption	Sql Injection	XSS	Directory Traversal	Http Response Splitting	Bypass something	Gain Information	Gain Privileges	CSRF	File Inclusion	# of exploits
2009	5	3								1					
2010	1	1	1												
2011	9	1	1		1					3	2	3			
2012	8	5	4	2							1				1
2013	7	1	2	2	2					1	1	3			
2014	13	2	4	1		1				1	2	2			1
2015	125	56	70	63	46					20	19	17			
2016	523	105	73	92	38					48	99	250			
2017	842	87	206	162	32			1		31	115	36			
2018	78	11	22	18	1					2	21	1			

Figure 1 – Android Vulnerability Statistics

The user’s engagement can be lowered if they are only presented with ideas and solutions throughout the usage of the app. As per the study conducted by Looyestyn et al. the gamification of applications are found to have a positive impact on the engagements in programs (Looyestyn et al., 2017). In another case study (Maria I et al., 2014) performed by Maria Ibanez based on computer science studies students in learning activities using gamification provides an intresting perspective. The study is based on student’s learning programming languages through score (badges) based activities. It’s concluded that the one of the main reasons for the students completing the activities is due to the desire of collecting all the badges. It’s also noted that having a leaderboard discouraged some students’ participation efforts. Neverthless Gamification in learning is found to have a positive effect on students’ engagement.

2.2 Background & Existing Implementations

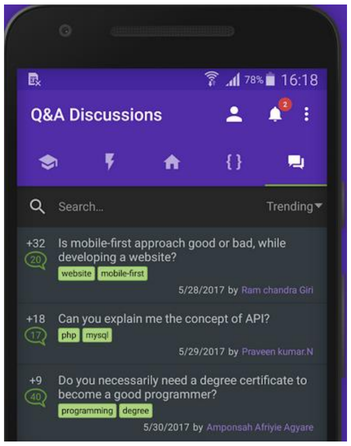
As Android development sector grows bigger every year, the need for security in the coding and design of an application is crucial in avoiding exploitable functions. Many bugs that are relatively easy to fix involves the prevention data leakage/storage of confidential information. The storage methods especially are to be designed with security as priority.

The Logging method is used by android store important notes during the runtime of the application. As noted in the official android developer documentation (Developers Android, 2018), the permission “READ_LOGS” allows apps within the device to read data from logs. This log system is prone to data leaks, if the developer records crucial data such as a password along the error logs, it could be exploited through malicious activities.

Applications that also employ a database are also prone to various methods of exploitation. The Android Operating system natively supports SQLite database implementation, which is vulnerable to injection attacks that expose database information. In applications that contains a SQLite database which aims to store private data of user (e.g. address, pass etc.), the risk of injection due to improper coding functions and error handling is high. Similar to other platforms the android programs that incorporate SQL database should follow proper techniques such as the usage of ‘Prepared Statements’ to avoid common unintended data retrievals.

There are few applications and implementations in the area of the android threat awareness. Most of the applications focus on educating the android app developers on the secure coding practices and using updated functions that are less prone to bugs. Their intentions are fulfilled by following a standard set of vulnerability types list as deemed popular by reputable foundations such as OWASP. OWASP’s “SeraphimDroid Project” (OWASP, 2015) aims to educate android users on the privacy and security threats. This program’s features include permission scanners that check permission of applications in a device. This feature can display any type of permission exploits that occur within the device. It’s also contains a service locking function that stops services that may harm or abuse the device. Services that are native or otherwise are blocked from exploiting data.

Sololearn (Sololearn, 2017) is one of the biggest learning portals that provides android users with coding lessons and activities. It has several features that simulate a learning environment for the users such as compiler for running small code. Discussions are one of the functionalities of the application that provides a community experience for the users. The Forum allows the users to discuss with each other around world on topics related to programming.



+32 20 Is mobile-first approach good or bad, while developing a website?
website mobile-first
5/28/2017 by Ram chandra Giri

+18 17 Can you explain me the concept of API?
php mysql
5/29/2017 by Praveen kumar.N

+9 40 Do you necessarily need a degree certificate to become a good programmer?
programming degree
5/30/2017 by Amponsah Afriyie Agyare

Another such application, 'damn vulnerable iOS application' (DVIA, 2015) looks at employing exploits within the application in order to teach users of threats and risks within apple devices. The application incorporates situations that highlight risk types as found on apps and among everyday devices.

2.3 Conclusion

In studying the existing implementations for mobile vulnerability awareness, it can be seen that there are features and room for improvement on such applications.

This project aims to create a learning environment that mixes aspects of fun and knowledge. While the solutions offered in terms of areas vulnerability area coverage, the elements of fun are lacking. This project application will focus on making the learning sessions a more enthralling experience for the users. This goal will be achieved by adding a 'gaming trait' of obtaining score/points for learning and solving bugs. With the addition of this trait, the users can learn and be better equipped in dealing with vulnerabilities while having an engaging involvement during usage of the application. Similar type of score systems are incorporated in Capture the Flag Tournament (CTF) events in the cyber security sector which have demonstrated the effectiveness of score/leader board functions. In addition, the biggest feature of the app will be the inclusion of a forum board. Registered users can login and create discussion on topics related to vulnerabilities where they can be discussed by other users.

3 Project Planning

3.1 Motivation

The idea chosen for the project is an android based vulnerability application. The need for security in the area of android programming is essential to say the least as the mobile application sector grows every year. As revenue generated from apps increase exponentially (Statista, 2018), the awareness for building and providing secure software and services for the market is required since they directly affect consumer's privacy or security. The project aims to spread awareness about risks involved in android application development through learning exercises. It focuses on delivering a community driven, challenge solving environment with addition of discussion forum. Taking the learning exercises (the concept) and combining it with elements of games (a Flag/scoring system etc.) similar to those found in Cyber Security Challenges (CTF) is the end goal. By achieving the combination of such concepts, it's possible to effectively educate developers on various vulnerabilities on different parts of an application.

3.2 Tasks Involved

Expected Results for the Project

- Creating a Vulnerability Application that works as intended.
- App should be polished and easy to use
- Should be relatively secure and bug free

Expected Work

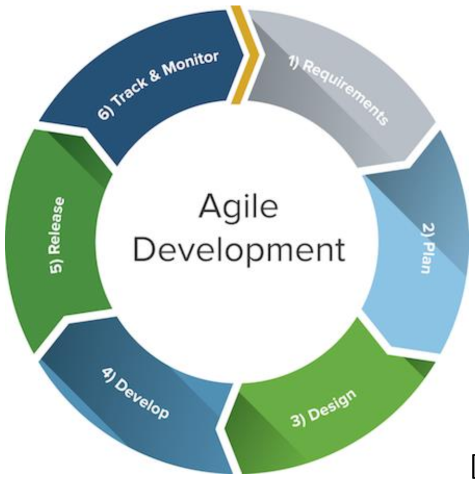
- Being able to work with Android OS and find vulnerabilities that occur due to programming.
- Being able to deliver learning exercises combined with score systems.
- Being able to deliver an interactive experience on application exploits.

Possible Difficulties

- Depth of Programming related to android OS may be huge.
- Barrier in learning for target audience about vulnerabilities due to limited knowledge on some areas.
- Unable to fulfil all the objectives of the application due to technical constraints.

3.5 Development Methodology – AGILE

Agile Development is the methodology that closely parallels this application's development cycle. This model consists of six different stages.



[Smartsheet, 2017]

Requirements
The requirements (develop a working vulnerability learning software) that needs to be fulfilled are outlined during this phase. Main objectives of the application and optional functions are decided.

Plan
This stage involves planning of the development strategies that will be utilized. Description of end product is drawn which composes of necessary functionalities and other work that are mandatory for core functionality of the application.

Design
Designs for early versions of the application are created. Requirements checklist is followed for this to construct designs for most functionalities.

Development
Design plans are materialized in this stage. Coding is carried out and most of the features required is built.

Testing & Release
The release version of the application is finalized. Testing for errors, bugs and security flaws are also performed. The app is presented to test groups to garner feedback to further improve its aspects. The application is released to the market.

Track and Monitor
This Phase is more or less an optional phase in our project. The released app is periodically monitored for bugs or issues based on user reviews.

Advantages and Disadvantages of this model

There are notable benefits when using this model:

- Ability to change certain design principle with ease, as the name suggests the cycle allows flexibility in work priorities (between vulnerability modules).

- Final product features are not set in stone. The final stages of the model, namely Track and Monitor allows to follow the progression of the applications usage and better optimize based on the recommendations from the consumers (adding/optimizing forum functionalities etc.).

As with any other development models there are cons for the agile model:

- Flexibility experienced while using the model, can cause deviations of the original vision of the project.

- Design and development stages can be intensive as they require proper principles to be implemented and expertise on areas that maybe less popular while the application is being coded.

4 System Requirements, Specification and Plans

4.1 Deliverables/Objectives

The Deliverables/objectives that are to be presented for the project is as follows:

- Create a plan for the project.
- Research and write about different parts of the project.
- Producing an analysis and design documentation.
- Testing and developing of model versions of the application.
- Including a literature review based on research.
- Produce a solutions documentation after the core application design.

4.2 Functional Requirements

Vulnerability Application

The core of the application will contain a set of challenges that introduces vulnerable designs within the application code. The challenges are presented in the form of scenarios that are provided to the user to solve or learn the fixes of certain exploits.

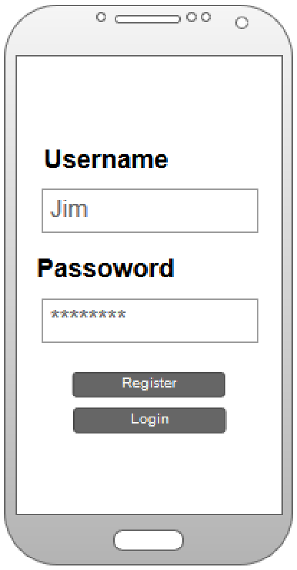
The main interface will provide a brief introduction to the users about the application. It will contain a link to scenarios function of the application. Scenarios interface will comprise of several exploits exercises that can be solved. The exploits presented in scenarios can ranging various difficulty and scope levels. Vulnerabilities can include a variety of areas including the exploitation of privilege elevation, insecure data storage and lack of proper input validation.

4.2.1 Diagrams

The concept diagrams for various features of the application as drawn using the software draw.io (draw.io, 2018) are as follows.

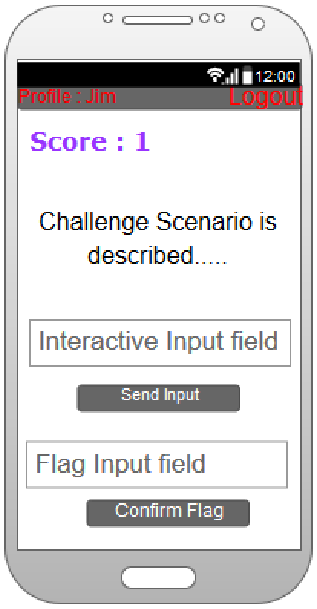
Registration and Login interface

This interface will contain functions for user registration and login.



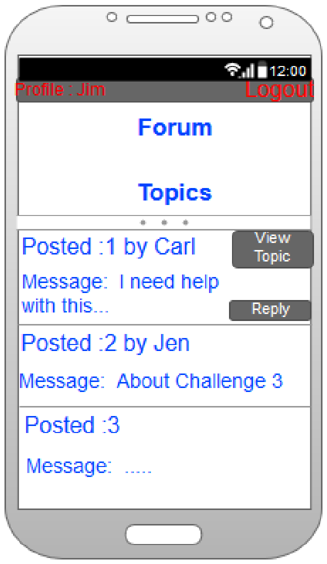
Challenges/Scenarios

The Scenarios/Challenges will be presented in the following interface. It will contain a score tracker, and description for a particular vulnerability scenario. It will also have a flag input field for earning scores.



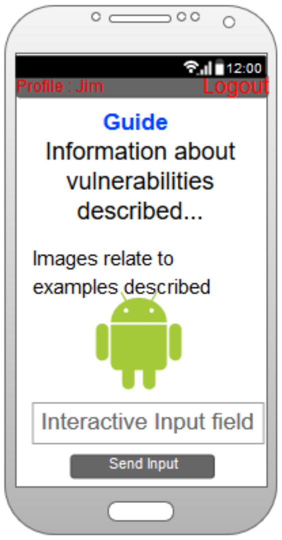
Forum Board

The forum board will contain, posts by users where other users can view their topics and reply to them



Guide

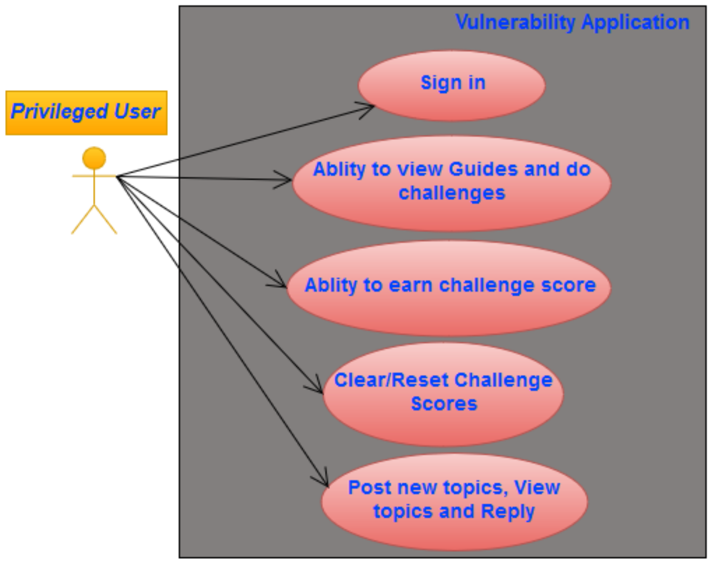
A simple guide interface will consists of descriptions of vulnerabilities or risks in programming. In some cases include interactive images or input fields & response messages for learning.


12:00
Profile : Jim
Logout
Guide
Information about vulnerabilities described...
Images relate to examples described
Interactive Input field
Send Input

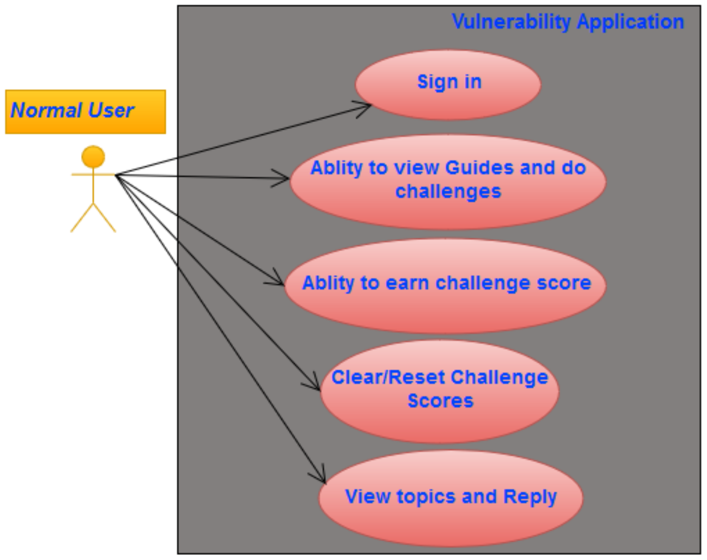
4.2.2 Use case Diagrams

The normal users have most of the features available for use, except for functions such as posting new topic.

Use case with user actions for special users.


Vulnerability Application
Privileged User
Sign in
Ablity to view Guides and do challenges
Ablity to earn challenge score
Clear/Reset Challenge Scores
Post new topics, View topics and Reply

Use case with user actions for regular users.


Vulnerability Application
Normal User
Sign in
Ablity to view Guides and do challenges
Ablity to earn challenge score
Clear/Reset Challenge Scores
View topics and Reply

4.3 Data Requirements

- Storage of Challenge Scores and making it persistent when the app is installed.
- Validation of inputs from exploit exercises and the flags.
- Retrieving Learning components to display guide on scenarios and fixes.
- Retrieving discussion board specific data, displaying threads and replies.

The SQLite database contains 2 different tables (one for user accounts & one for a challenge).

Database table contents/structure is separated into three different columns: id as the primary unique identifier, username and password as the coordinates to be stored.

SQLite database for users contain the users table which holds user account data.

Column Name	**Description**	**Data type**
id(Primary Key)	an unique id that is present for identifying an user	int
username(only Distinct values)	Username values chosen by registered users.	string
password(only Distinct values)	Passwords chosen by registered users	string

Shared Preference key values which are used for local storage of the various types of data.

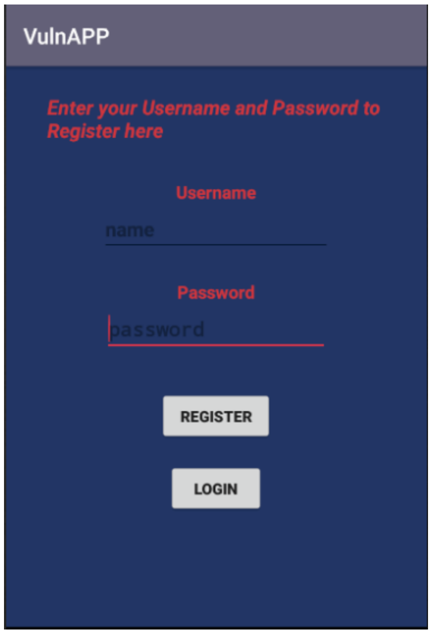
Share Preference	**Key**	**Description**	**Data type**
MyPref	scoreforid	A key to store user scores.	int
MyPref	threaddet	Store thread/topic replies	string
MyPref	postdet	Store post details	string
MyPref	nameStr	Store logged in user’s name(temporarily)	string

5 System Design & Implementation

5.1 Android Application Implementation

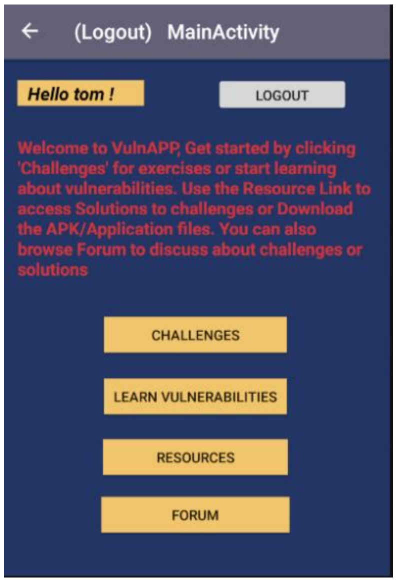
Register & Login

When the application is launched, the user is presented with a register system, where they can create an account. It also provides a login button after the user is registered.



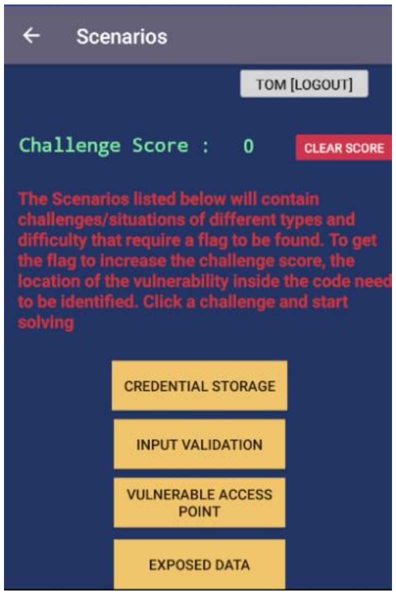
Welcome/Main Screen

The Welcome screen is then displayed after login. It contains a brief description of the app and four buttons (Learn Vulnerabilities, Challenges, Resources, Forum) that links to the apps features. The user logged is also indicated along with a logout button on top of the interface.



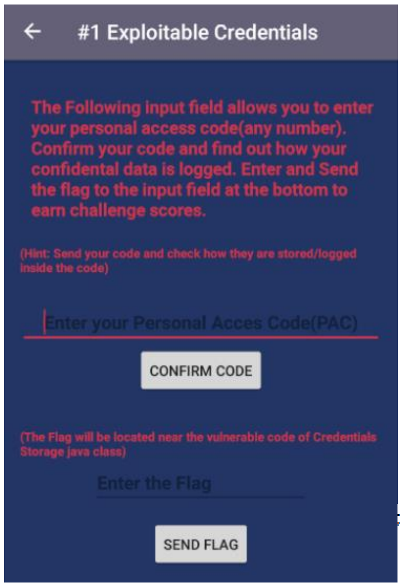
Welcome to VulnAPP, Get started by clicking 'Challenges' for exercises or start learning about vulnerabilities. Use the Resource Link to access Solutions to challenges or Download the APK/Application files. You can also browse Forum to discuss about challenges or solutions

The Scenarios interface will contain all the exploit challenges. It also displays a challenge score system that keeps track of user's achievements in solving exercises. The Scores can be cleared anytime using clear score button. When the user selects one of the scenarios, the exploit screen is presented to them. Users are briefed on the situation surrounding the challenge (e.g. Vulnerable Search Function for a book library) and are requested to find the vulnerability in the code/application. All of the challenges will contain a flag near the vulnerable code or produces a result with a flag, when the user exploits the inputs. For every correct flag obtained, the score increases (as show in the Scenarios screen).



The Scenarios listed below will contain challenges/situations of different types and difficulty that require a flag to be found. To get the flag to increase the challenge score, the location of the vulnerability inside the code need to be identified. Click a challenge and start solving

Exploitable Credentials is one of the challenge that tests user's knowledge regarding unsafe storage in application. It explains a scenario where exploitable confidential information is stored. It accepts input and requires users to find the vulnerable location in the code. The flag input at the bottom provides a score if the correct flag (found near the code) is entered.



#1 Exploitable Credentials

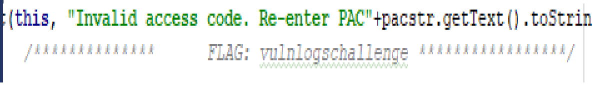
The Following input field allows you to enter your personal access code(any number). Confirm your code and find out how your confidental data is logged. Enter and Send the flag to the input field at the bottom to earn challenge scores.

(Hint: Send your code and check how they are stored/logged inside the code)

Enter your Personal Acces Code(PAC)

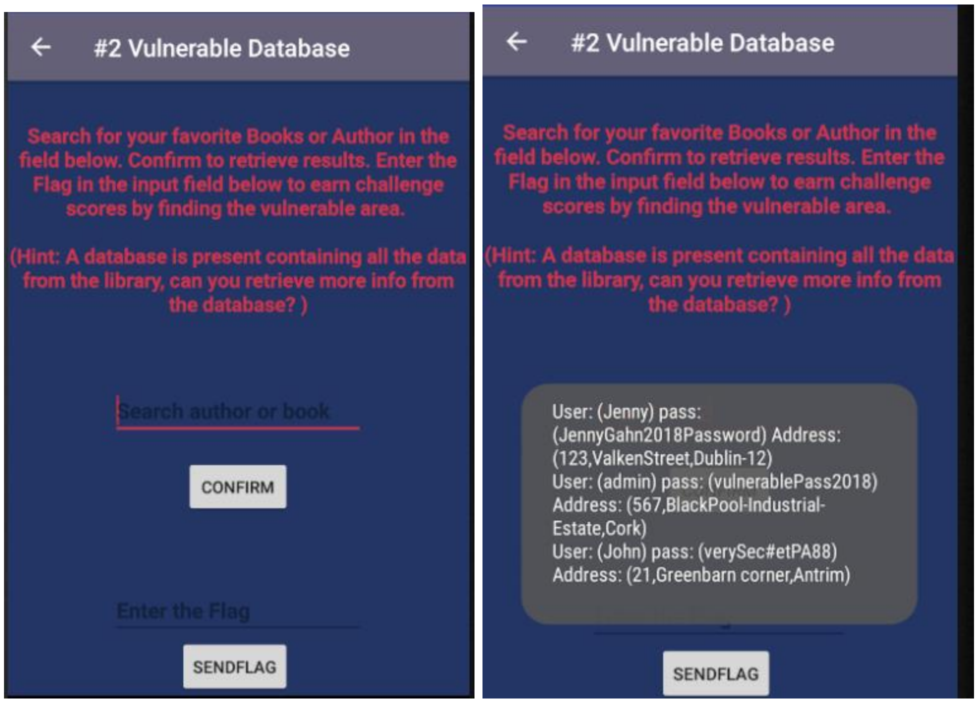
(The Flag will be located near the vulnerable code of Credentials Storage java class)

Enter the Flag



```
(this, "Invalid access code. Re-enter PAC"+pacstr.getText().toStrin
 /**************        FLAG: vulnlogschallenge ****************/
```

Challenge #2 also follows similar format to the previous exercise. It require users to input data into the first field which retrieves data from the database. User is encouraged to gather crucial data from the DB. The flag for the score, however is found in the response to the user search input.



#2 Vulnerable Database

Search for your favorite Books or Author in the field below. Confirm to retrieve results. Enter the Flag in the input field below to earn challenge scores by finding the vulnerable area.

(Hint: A database is present containing all the data from the library, can you retrieve more info from the database?)

Search author or book

Enter the Flag

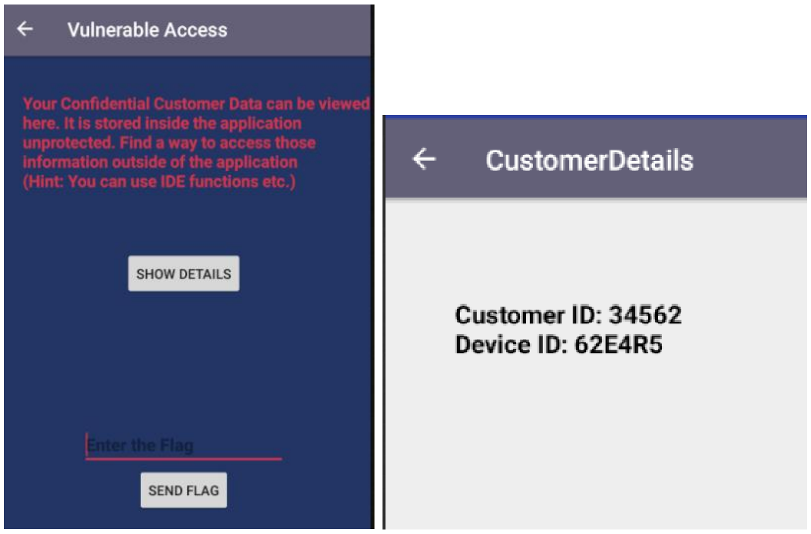
#2 Vulnerable Database

Search for your favorite Books or Author in the field below. Confirm to retrieve results. Enter the Flag in the input field below to earn challenge scores by finding the vulnerable area.

(Hint: A database is present containing all the data from the library, can you retrieve more info from the database?)

```
User: (Jenny) pass:
(JennyGahn2018Password) Address:
(123,ValkenStreet,Dublin-12)
User: (admin) pass: (vulnerablePass2018)
Address: (567,BlackPool-Industrial-
Estate,Cork)
User: (John) pass: (verySec#etPA88)
Address: (21,Greenbarn corner,Antrim)
```

Vulnerable access is one of the challenges that require retrieving some data. The user is presented with the data when they view the customer details interface. They are then required to obtain this information outside of the app.



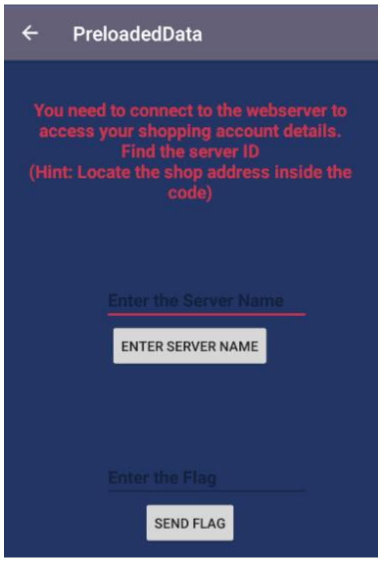
Vulnerable Access

Your Confidential Customer Data can be viewed here. It is stored inside the application unprotected. Find a way to access those information outside of the application (Hint: You can use IDE functions etc.)

CustomerDetails

Customer ID: 34562
Device ID: 62E4R5

Preloaded data challenge focuses on the issues of coding confidential values into the application. The goal of this challenge is to find the server ID inside the code and the flag near the ID.

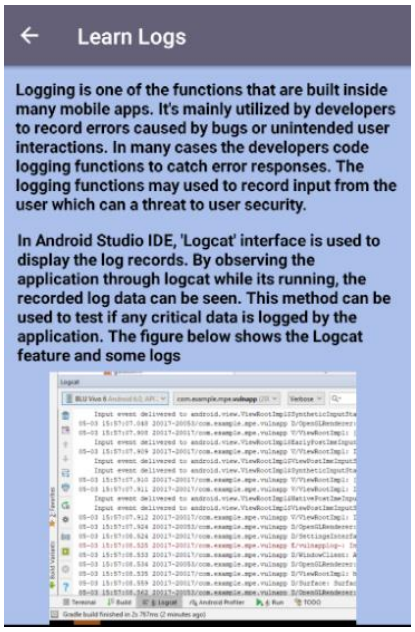


PreloadedData

You need to connect to the webserver to access your shopping account details. Find the server ID (Hint: Locate the shop address inside the code)

Learn Vulnerabilities

Learning about logs is one of the sections dedicated to logging functions in android development. It provides information about how logging is used and it also shows how logs can be viewed for debugging purposes. The learning sample directly correlates with challenge for storage vulnerabilities.



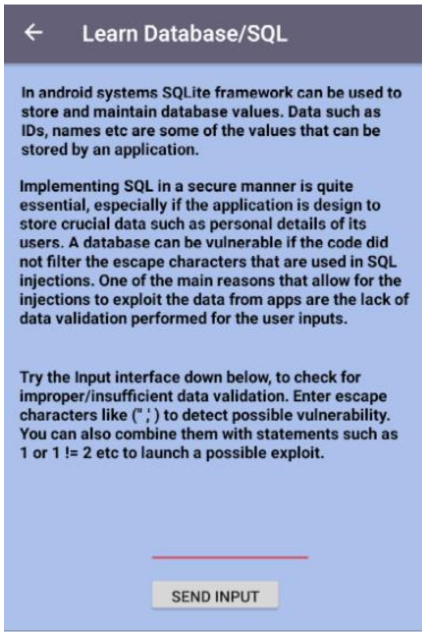
Logging is one of the functions that are built inside many mobile apps. It's mainly utilized by developers to record errors caused by bugs or unintended user interactions. In many cases the developers code logging functions to catch error responses. The logging functions may used to record input from the user which can a threat to user security.

In Android Studio IDE, 'Logcat' interface is used to display the log records. By observing the application through logcat while its running, the recorded log data can be seen. This method can be used to test if any critical data is logged by the application. The figure below shows the Logcat feature and some logs

```
     Input event delivered to android.view.ViewRootImpl$SyntheticInputSta
05-03 15:57:07.048 20017-20053/com.example.mpe.vulnapp D/OpenGLRenderer:
05-03 15:57:07.908 20017-20017/com.example.mpe.vulnapp V/ViewRootImpl:
     Input event delivered to android.view.ViewRootImpl$EarlyPostImeInput
05-03 15:57:07.909 20017-20017/com.example.mpe.vulnapp V/ViewRootImpl: I
     Input event delivered to android.view.ViewRootImpl$ViewPostImeInput
     Input event delivered to android.view.ViewRootImpl$SyntheticInputSta
05-03 15:57:07.910 20017-20017/com.example.mpe.vulnapp V/ViewRootImpl: I
05-03 15:57:07.911 20017-20017/com.example.mpe.vulnapp V/ViewRootImpl: I
     Input event delivered to android.view.ViewRootImpl$NativePostImeInpu
     Input event delivered to android.view.ViewRootImpl$ViewPostImeInput
05-03 15:57:07.912 20017-20017/com.example.mpe.vulnapp V/ViewRootImpl: I
05-03 15:57:07.924 20017-20053/com.example.mpe.vulnapp D/OpenGLRenderer:
05-03 15:57:08.524 20017-20017/com.example.mpe.vulnapp D/SettingsInterfa
05-03 15:57:08.525 20017-20017/com.example.mpe.vulnapp E/vulnapplog-: In
05-03 15:57:08.533 20017-20017/com.example.mpe.vulnapp D/WindowClient: A
05-03 15:57:08.534 20017-20053/com.example.mpe.vulnapp D/OpenGLRenderer:
05-03 15:57:08.535 20017-20017/com.example.mpe.vulnapp D/ViewRootImpl: M
05-03 15:57:08.559 20017-20017/com.example.mpe.vulnapp D/Surface: Surfac
05-03 15:57:08.562 20017-20053/com.example.mpe.vulnapp D/OpenGLRenderer:
```

Gradle build finished in 2s 767ms (2 minutes ago)

Another Learning material involves the database creation/function found in android applications. It provides an overview on technology implemented in databases and how exploitable they can be if not properly executed.

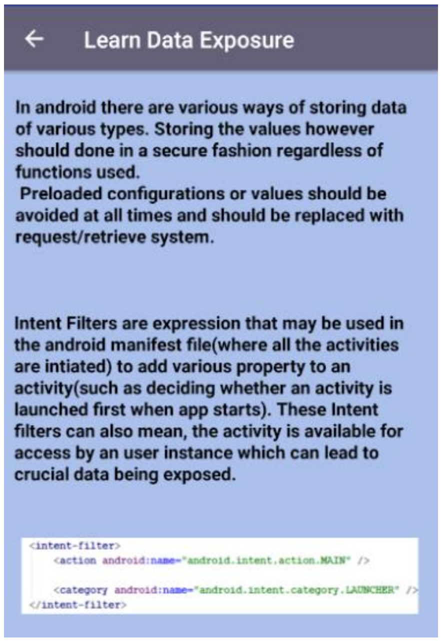


In android systems SQLite framework can be used to store and maintain database values. Data such as IDs, names etc are some of the values that can be stored by an application.

Implementing SQL in a secure manner is quite essential, especially if the application is design to store crucial data such as personal details of its users. A database can be vulnerable if the code did not filter the escape characters that are used in SQL injections. One of the main reasons that allow for the injections to exploit the data from apps are the lack of data validation performed for the user inputs.

Try the Input interface down below, to check for improper/insufficient data validation. Enter escape characters like (" ;) to detect possible vulnerability. You can also combine them with statements such as 1 or 1 != 2 etc to launch a possible exploit.

Data Exposure is one of the risks that is addressed in the 'Learn Data Exposure' section it explains common cirucmstances where data can be at a risk of exposure.



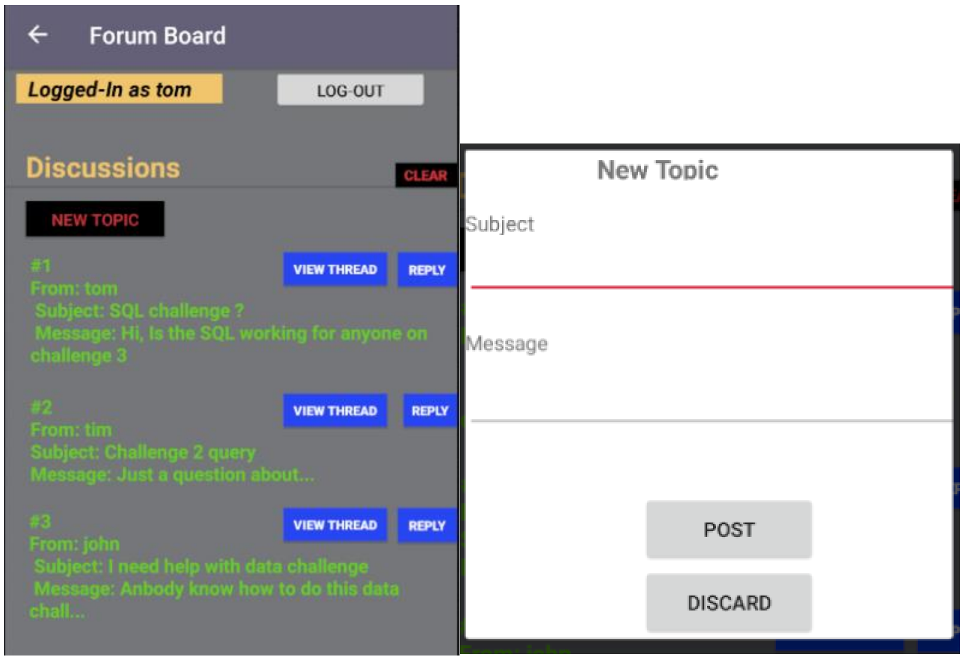
In android there are various ways of storing data of various types. Storing the values however should done in a secure fashion regardless of functions used.
Preloaded configurations or values should be avoided at all times and should be replaced with request/retrieve system.

Intent Filters are expression that may be used in the android manifest file(where all the activities are intiated) to add various property to an activity(such as deciding whether an activity is launched first when app starts). These Intent filters can also mean, the activity is available for access by an user instance which can lead to crucial data being exposed.

```
<intent-filter>
    <action android:name="android.intent.action.MAIN" />

    <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

Forum Board

The Forum interface serves as a discussion portal for the users on the challenges and vulnerabilities. The user logged in displayed at the top. In the discussion section, the Members who are provided special privileges can start new topics and reply. Each of those members have their own "talk space" where they can share their queries or other information. Other members are restricted from starting topics but can view the threads started by the special users and reply to their topics. Each thread/topic will contain the message origination profile, subject of the topic and the main message. Thread View allows user to read replies to a particular discussion. "View thread" displays information such as the identity of the users who replied and their messages.

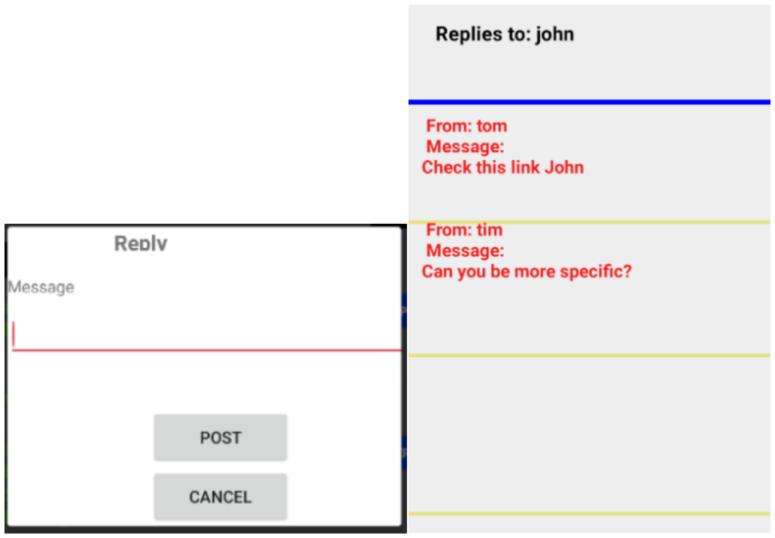


#1
From: tom
Subject: SQL challenge ?
Message: Hi, Is the SQL working for anyone on challenge 3

#2
From: tim
Subject: Challenge 2 query
Message: Just a question about...

#3
From: john
Subject: I need help with data challenge
Message: Anbody know how to do this data chall...

The Reply function allows any registered users to post messages to any topics/thread. The replies will then be posted to the thread space where they can viewed and replied to.

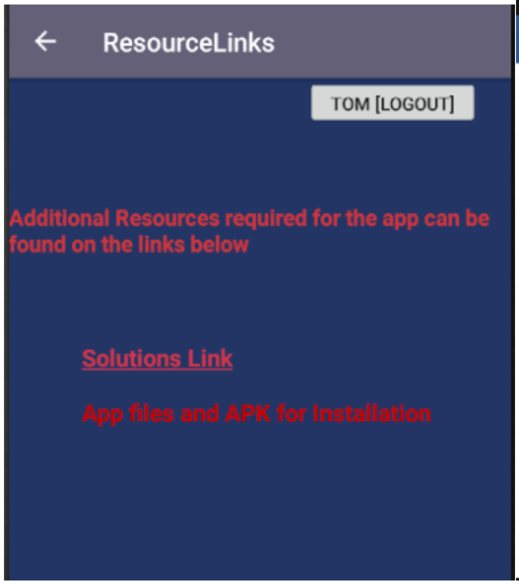


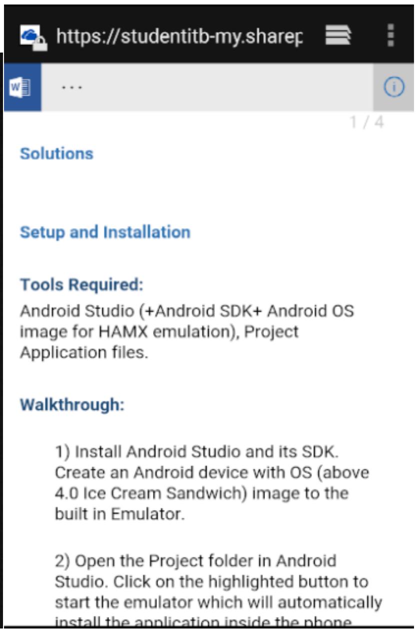
From: tom
Message:
Check this link John

From: tim
Message:
Can you be more specific?

Resource Links

The resource links interface allows users to access solutions to the challenges (Solutions, 2018), application files and APK are also available. Other Relevant resources will also be available for the users.


ResourceLinks
TOM [LOGOUT]
Additional Resources required for the app can be found on the links below
Solutions Link
App files and APK for Installation



Solutions

Setup and Installation

Tools Required:
Android Studio (+Android SDK+ Android OS image for HAMX emulation), Project Application files.

Walkthrough:

1) Install Android Studio and its SDK. Create an Android device with OS (above 4.0 Ice Cream Sandwich) image to the built in Emulator.

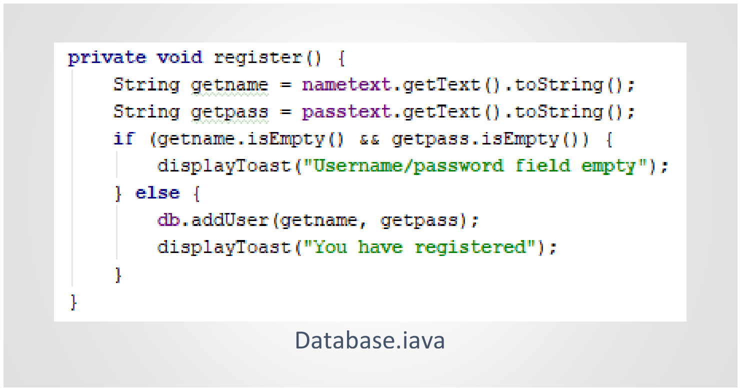
2) Open the Project folder in Android Studio. Click on the highlighted button to start the emulator which will automatically install the application inside the phone

5.2 Functional Design and Code Structure Walkthrough

The following section describes the code design and functionalities.

Register, Login Function & Database

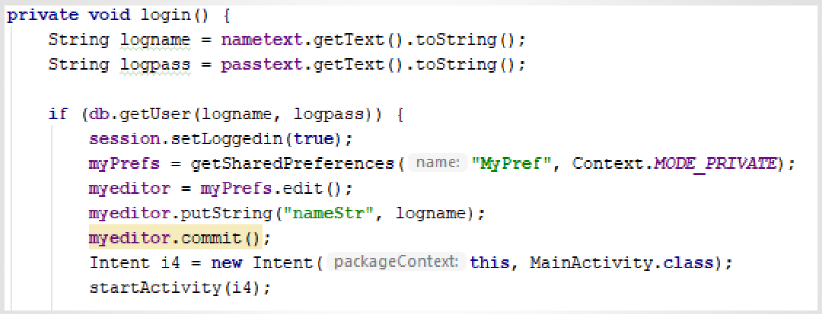
When the app is launched for the first time, the user is requested to register. The registration of the users are handled by the register method inside the Register.java class. The username and the passwords data are received from the user input and are passed into the database for a new ID creation.



```
private void register() {
    String getname = nametext.getText().toString();
    String getpass = passtext.getText().toString();
    if (getname.isEmpty() && getpass.isEmpty()) {
        displayToast("Username/password field empty");
    } else {
        db.addUser(getname, getpass);
        displayToast("You have registered");
    }
}
```

Database.iava

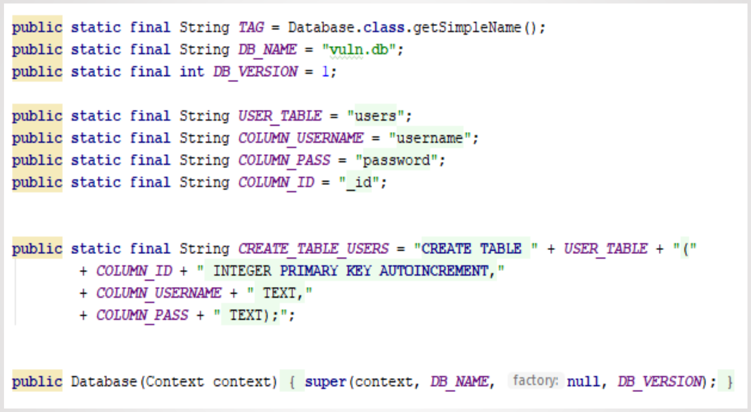
The Login function is handled by the login method, which retrieves user data from the database. If the input sent from the user matches the data values from the database, the user is granted a successful login. The logged in username is recoded with the use of ‘shared preferences’ for tracking the log status of the user later on.



```java
private void login() {
    String logname = nametext.getText().toString();
    String logpass = passtext.getText().toString();

    if (db.getUser(logname, logpass)) {
        session.setLoggedin(true);
        myPrefs = getSharedPreferences( name: "MyPref", Context.MODE_PRIVATE);
        myeditor = myPrefs.edit();
        myeditor.putString("nameStr", logname);
        myeditor.commit();
        Intent i4 = new Intent( packageContext: this, MainActivity.class);
        startActivity(i4);
```

Database.java

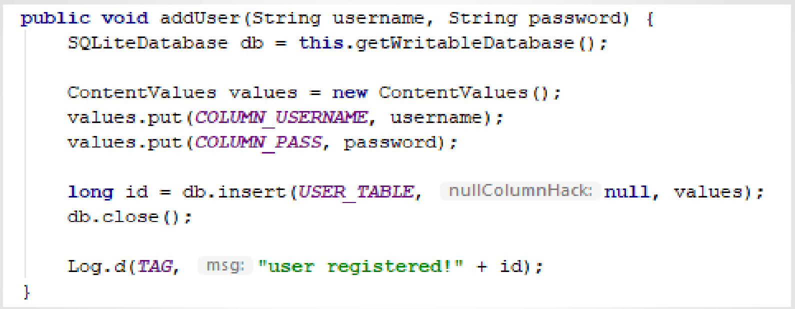
Database.java class handles the creation and other functions of user database using SQLite DB management. The columns for users, passwords and IDs are created for the “users” table.



```java
public static final String TAG = Database.class.getSimpleName();
public static final String DB_NAME = "vuln.db";
public static final int DB_VERSION = 1;

public static final String USER_TABLE = "users";
public static final String COLUMN_USERNAME = "username";
public static final String COLUMN_PASS = "password";
public static final String COLUMN_ID = "_id";


public static final String CREATE_TABLE_USERS = "CREATE TABLE " + USER_TABLE + "("
        + COLUMN_ID + " INTEGER PRIMARY KEY AUTOINCREMENT,"
        + COLUMN_USERNAME + " TEXT,"
        + COLUMN_PASS + " TEXT);";


public Database(Context context) { super(context, DB_NAME,  factory: null, DB_VERSION); }
```

Database.java

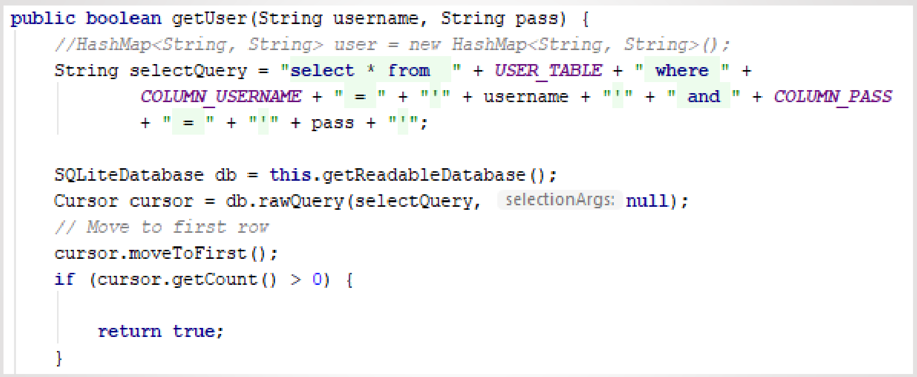
The addUser method adds new ID to the user table every time a registration occurs.



```java
public void addUser(String username, String password) {
    SQLiteDatabase db = this.getWritableDatabase();

    ContentValues values = new ContentValues();
    values.put(COLUMN_USERNAME, username);
    values.put(COLUMN_PASS, password);

    long id = db.insert(USER_TABLE,  nullColumnHack: null, values);
    db.close();

    Log.d(TAG,  msg: "user registered!" + id);
}
```

Database.java

The SQLite retrieve function is performed by getUser method which fetches user values for logins.



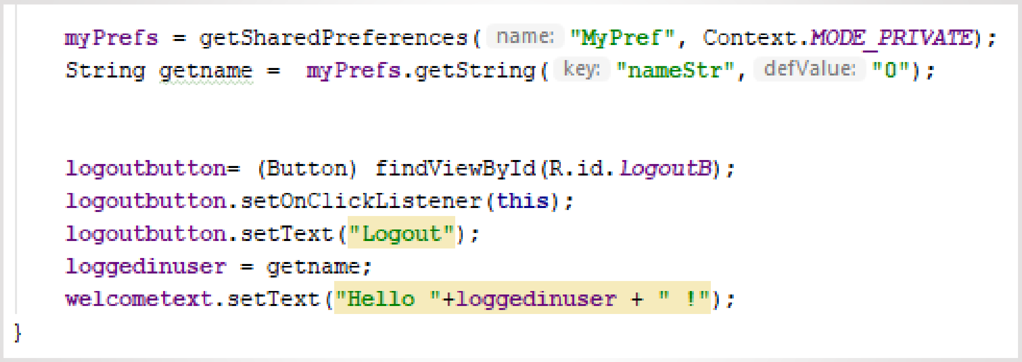
```
public boolean getUser(String username, String pass) {
    //HashMap<String, String> user = new HashMap<String, String>();
    String selectQuery = "select * from  " + USER_TABLE + " where " +
            COLUMN_USERNAME + " = " + "'" + username + "'" + " and " + COLUMN_PASS
            + " = " + "'" + pass + "'";

    SQLiteDatabase db = this.getReadableDatabase();
    Cursor cursor = db.rawQuery(selectQuery, selectionArgs: null);
    // Move to first row
    cursor.moveToFirst();
    if (cursor.getCount() > 0) {

        return true;
    }
```

Database.java

Main Screen

When the user logins, they are presented the main menu of the application. From this screen they are provided with various button links to different sections of the app. MainActivity.java class contains the functions for creating the main interface.

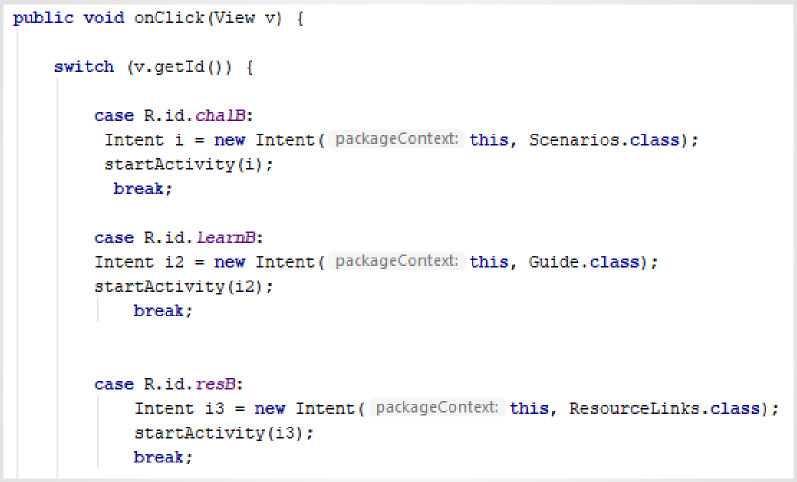
The logged in users name is retrieved in a key ("nameStr") which is used to display the log status of the user. The logout button is also placed alongside the status.



```
    myPrefs = getSharedPreferences( name: "MyPref", Context.MODE_PRIVATE);
    String getname =  myPrefs.getString( key: "nameStr", defValue: "0");


    logoutbutton= (Button) findViewById(R.id.logoutB);
    logoutbutton.setOnClickListener(this);
    logoutbutton.setText("Logout");
    loggedinuser = getname;
    welcometext.setText("Hello "+loggedinuser + " !");
}
```

MainActivity.java

The buttons links for all sections are equipped with 'button listeners' and the actions for each button is processed under "onClick" method. For each cases inside the method's switch statement, the corresponding activity/feature is launched.

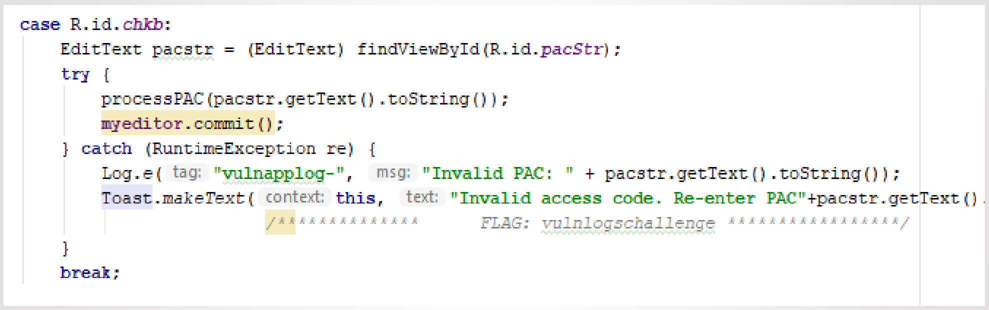


```java
public void onClick(View v) {

    switch (v.getId()) {

        case R.id.chalB:
         Intent i = new Intent( packageContext: this, Scenarios.class);
         startActivity(i);
          break;

        case R.id.learnB:
        Intent i2 = new Intent( packageContext: this, Guide.class);
        startActivity(i2);
            break;


        case R.id.resB:
            Intent i3 = new Intent( packageContext: this, ResourceLinks.class);
            startActivity(i3);
            break;
```

MainActivity.java

Credential storage

CredentialStorage.java class handles the design for one of the challenges presented in the application. The core functions inside the challenge is based around storing user input, when the user enters data into the requested fields. When the send button (chkb) is pressed, the input is retrieved from the text field and are logged using the Log.e function.

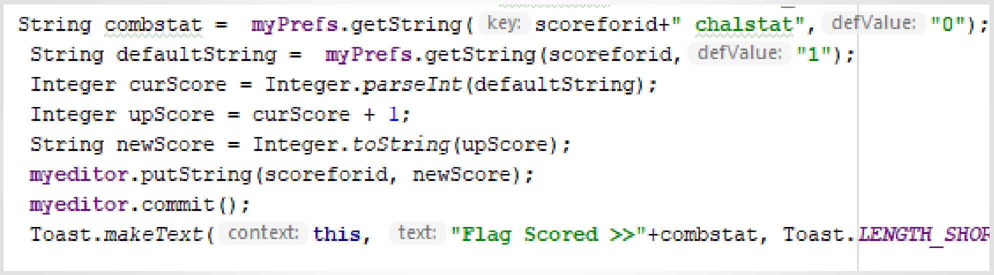


```java
case R.id.chkb:
    EditText pacstr = (EditText) findViewById(R.id.pacStr);
    try {
        processPAC(pacstr.getText().toString());
        myeditor.commit();
    } catch (RuntimeException re) {
        Log.e( tag: "vulnapplog-",  msg: "Invalid PAC: " + pacstr.getText().toString());
        Toast.makeText( context: this,  text: "Invalid access code. Re-enter PAC"+pacstr.getText().
                        /**************       FLAG: vulnlogschallenge *****************/
    }
    break;
```

CredentialStorage.java

An alert (toast) message is displayed to the user requesting them to enter again (mimicking a real-life design) while the confidential information is logged.

If the user is able to locate the Flag after the inspection of vulnerable code, they are prompted to enter the flag into the flag input field. If the flag entered matches the flag found, an if statement is used to process further actions.



```
String combstat =  myPrefs.getString( key: scoreforid+" chalstat", defValue: "0");
 String defaultString =  myPrefs.getString(scoreforid, defValue: "1");
 Integer curScore = Integer.parseInt(defaultString);
 Integer upScore = curScore + 1;
 String newScore = Integer.toString(upScore);
 myeditor.putString(scoreforid, newScore);
 myeditor.commit();
 Toast.makeText( context: this,  text: "Flag Scored >>"+combstat, Toast.LENGTH_SHOR
```

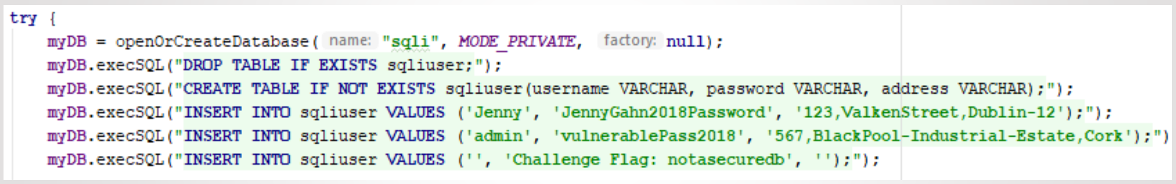
CredentialStorage.java

Inside the condition of flag being correct, the ‘scoredforid’ (the key responsible for storing the user’s challenge scores) shared preference key is retrieved. The score number from the key is then accessed and appends a score to the current score

Vulnerable Database

The java class VulnerableDatabase revolves around the challenge of exploiting an unsafe database.

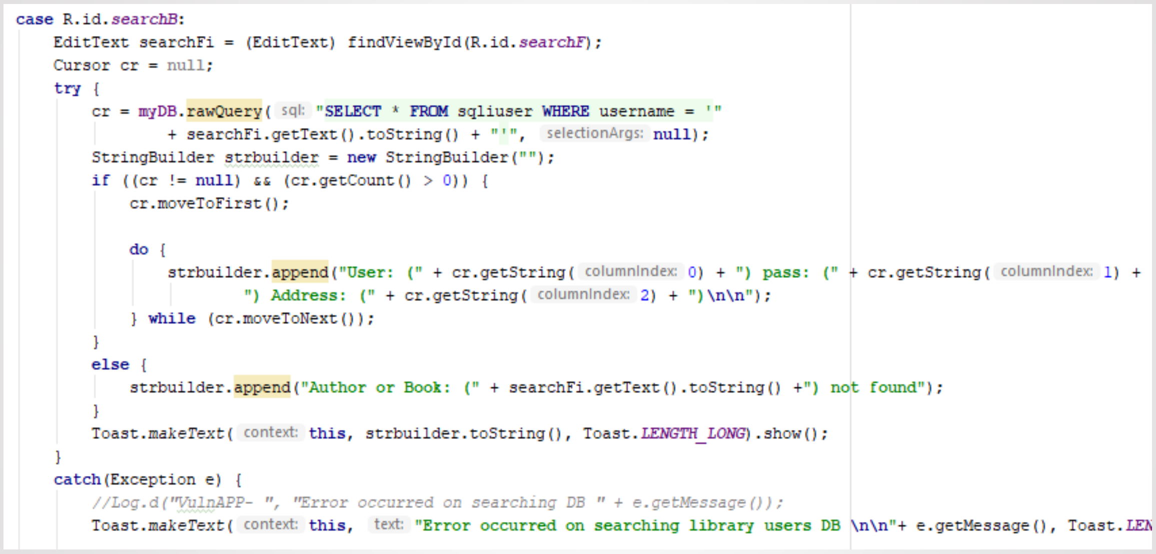
A database which stores user account information is created using SQLite. Data such as name, password address of the users are assigned to columns of the user table. It also contains a row with the challenge flag inside.



```
try {
    myDB = openOrCreateDatabase( name: "sqli", MODE_PRIVATE,  factory: null);
    myDB.execSQL("DROP TABLE IF EXISTS sqliuser;");
    myDB.execSQL("CREATE TABLE IF NOT EXISTS sqliuser(username VARCHAR, password VARCHAR, address VARCHAR);");
    myDB.execSQL("INSERT INTO sqliuser VALUES ('Jenny', 'JennyGahn2018Password', '123,ValkenStreet,Dublin-12');");
    myDB.execSQL("INSERT INTO sqliuser VALUES ('admin', 'vulnerablePass2018', '567,BlackPool-Industrial-Estate,Cork');")
    myDB.execSQL("INSERT INTO sqliuser VALUES ('', 'Challenge Flag: notasecuredb', '');");
```

VulnerableDatabase.java

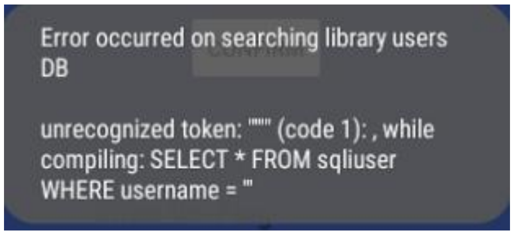
When the user searches for a book/author, the data entered into the input field is sent to the database query. The "Cursor" function which processes database activities is used in conjunction with the "rawQuery" function (searches for the values in the database from the user input. The input is then searched across the users database without any form of validation.



```
case R.id.searchB:
    EditText searchFi = (EditText) findViewById(R.id.searchF);
    Cursor cr = null;
    try {
        cr = myDB.rawQuery( sql: "SELECT * FROM sqliuser WHERE username = '"
                + searchFi.getText().toString() + "'", selectionArgs: null);
        StringBuilder strbuilder = new StringBuilder("");
        if ((cr != null) && (cr.getCount() > 0)) {
            cr.moveToFirst();

            do {
                strbuilder.append("User: (" + cr.getString( columnIndex: 0) + ") pass: (" + cr.getString( columnIndex: 1) +
                        ") Address: (" + cr.getString( columnIndex: 2) + ")\n\n");
            } while (cr.moveToNext());
        }
        else {
            strbuilder.append("Author or Book: (" + searchFi.getText().toString() +") not found");
        }
        Toast.makeText( context: this, strbuilder.toString(), Toast.LENGTH_LONG).show();
    }
    catch(Exception e) {
        //Log.d("VulnAPP- ", "Error occurred on searching DB " + e.getMessage());
        Toast.makeText( context: this,  text: "Error occurred on searching library users DB \n\n"+ e.getMessage(), Toast.LEN
```

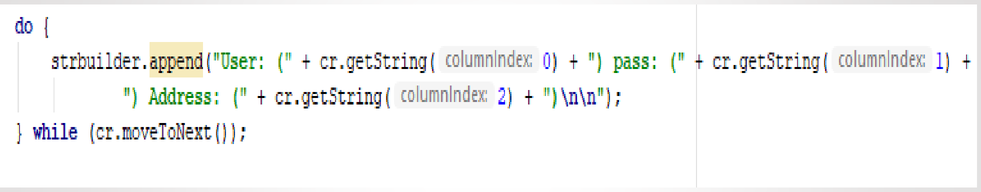
VulnerableDatabase.java

The error messages are displayed to users which will inform the users of the database structure if the value is not found in the database



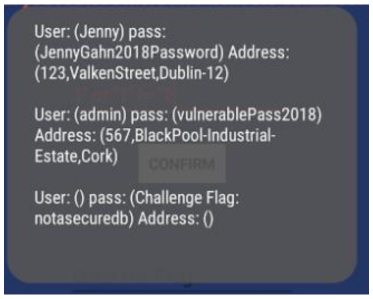
```
Error occurred on searching library users
DB

unrecognized token: "'''" (code 1): , while
compiling: SELECT * FROM sqliuser
WHERE username = ''
```

If the user manages to successfully input a query, a message is displayed along with the retrieved values from the database including the flag.



```
do {
    strbuilder.append("User: (" + cr.getString( columnIndex: 0) + ") pass: (" + cr.getString( columnIndex: 1) +
            ") Address: (" + cr.getString( columnIndex: 2) + ")\n\n");
} while (cr.moveToNext());
```

VulnerableDatabase.java

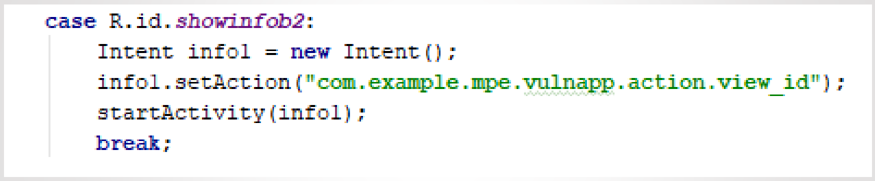


```
User: (Jenny) pass:
(JennyGahn2018Password) Address:
(123,ValkenStreet,Dublin-12)

User: (admin) pass: (vulnerablePass2018)
Address: (567,BlackPool-Industrial-
Estate,Cork)

User: () pass: (Challenge Flag:
notasecuredb) Address: ()
```

The challenge scores are also changed using similar methods as found in the previous challenge once the user inputs the correct flag into the flag field.

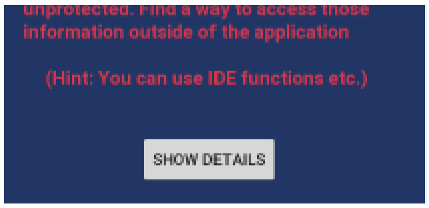
Vulnerable Access

The interface for this challenge contains a description A new intent action is set if the user clicks the show details button



```
case R.id.showinfob2:
    Intent infol = new Intent();
    infol.setAction("com.example.mpe.vulnapp.action.view_id");
    startActivity(infol);
    break;
```

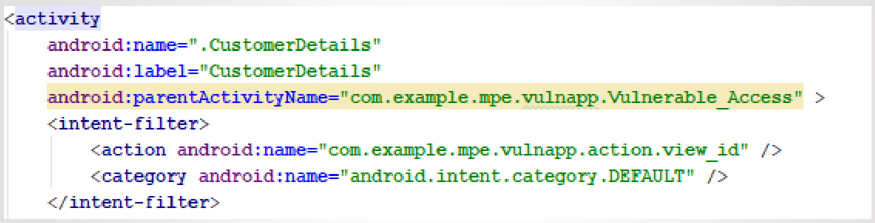
VulnerableAccess.java



unprotected. Find a way to access those information outside of the application

(Hint: You can use IDE functions etc.)

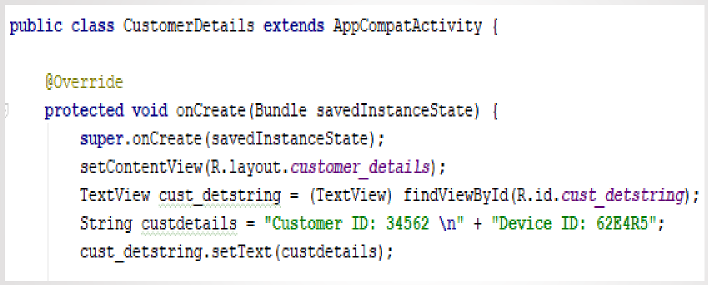
The intent then accesses the property (view_id) set on the androidmanifest.xml file which is assigned to launch the customerdetails activity



```
<activity
    android:name=".CustomerDetails"
    android:label="CustomerDetails"
    android:parentActivityName="com.example.mpe.vulnapp.Vulnerable_Access" >
    <intent-filter>
        <action android:name="com.example.mpe.vulnapp.action.view_id" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
```

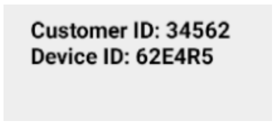
AndroidManifest.xml

The customer details activity displays a screen with thet textview containing the customer details and device ID.



```
public class CustomerDetails extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.customer_details);
        TextView cust_detstring = (TextView) findViewById(R.id.cust_detstring);
        String custdetails = "Customer ID: 34562 \n" + "Device ID: 62E4R5";
        cust_detstring.setText(custdetails);
```

CustomerDetails.java

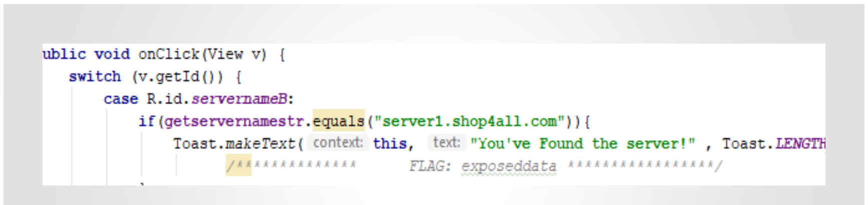


```
Customer ID: 34562
Device ID: 62E4R5
```

The new scores for that user are calculated and processed similar to the other challenges.

Preloaded Data

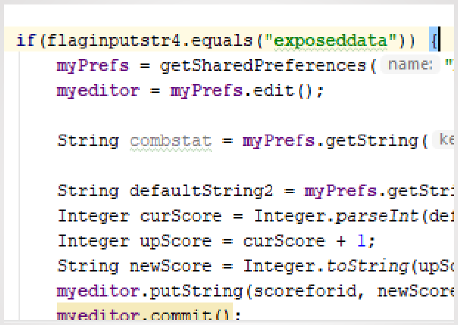
The PreloadedData.java file contains a button click switch case which performs input validation for the server name entered by the user. The server name is located in the button, the flag is also located close to validation code.



```
ublic void onClick(View v) {
    switch (v.getId()) {
        case R.id.servernameB:
            if(getservernamestr.equals("server1.shop4all.com")){
                Toast.makeText( context: this,  text: "You've Found the server!" , Toast.LENGTH
                        /**************        FLAG: exposeddata ****************/
```

PreloadedData.java

If the user finds the flag and submits it into the flag field, the user is awarded a score.

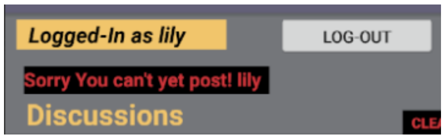


```
if(flaginputstr4.equals("exposeddata")) {
    myPrefs = getSharedPreferences( name: "
    myeditor = myPrefs.edit();

    String combstat = myPrefs.getString( ke

    String defaultString2 = myPrefs.getStr
    Integer curScore = Integer.parseInt(de
    Integer upScore = curScore + 1;
    String newScore = Integer.toString(upS
    myeditor.putString(scoreforid, newScore
    myeditor.commit();
```

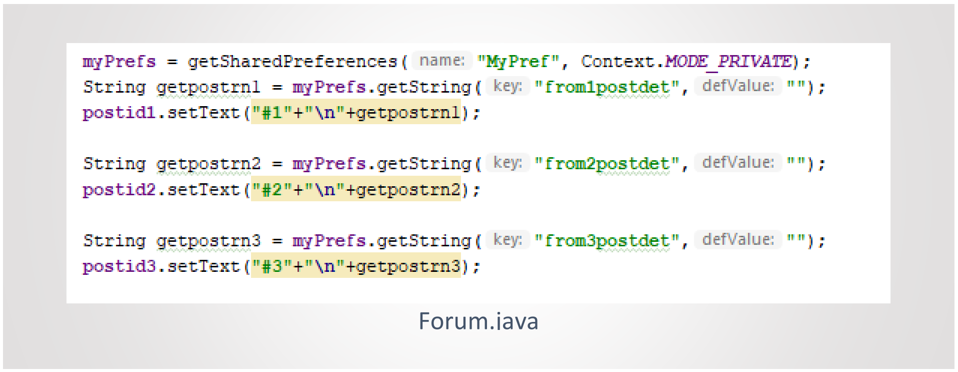
PreloadedData.java

Forum functions

When a user visits the forum, their profile is checked for privileges. If they the user doesn't have permission, a message is displayed which alerts them and the 'newtopic' button is rendered inaccessible



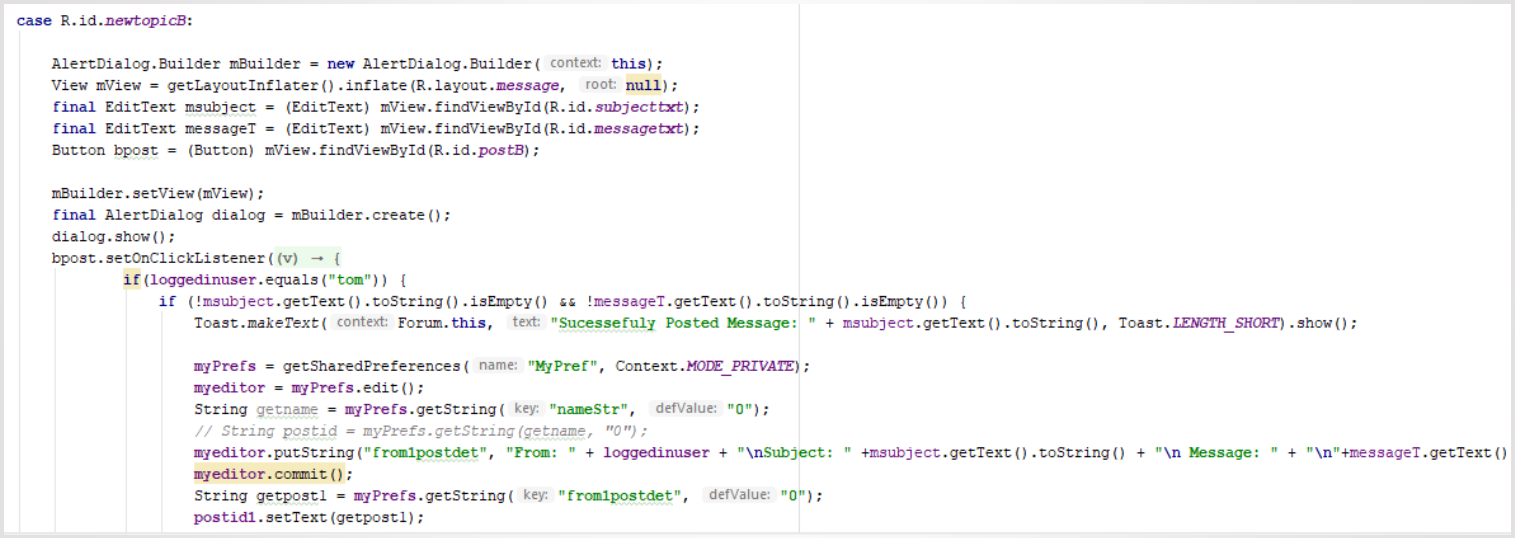
Topics created by the users are stored in shared preferences, which holds all the post details of users. When a user visits the forum the post are fetched from the shared preferences and displayed onto the discussion space.



```
myPrefs = getSharedPreferences( name: "MyPref", Context.MODE_PRIVATE);
String getpostrn1 = myPrefs.getString( key: "from1postdet", defValue: "");
postid1.setText("#1"+"\n"+getpostrn1);

String getpostrn2 = myPrefs.getString( key: "from2postdet", defValue: "");
postid2.setText("#2"+"\n"+getpostrn2);

String getpostrn3 = myPrefs.getString( key: "from3postdet", defValue: "");
postid3.setText("#3"+"\n"+getpostrn3);
```

Forum.iava

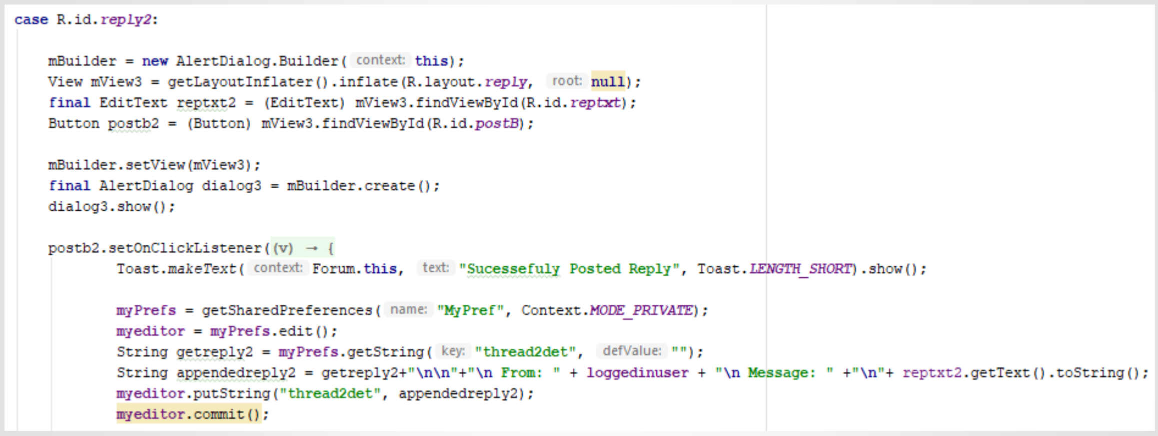
When the new topic button is pressed a post dialog box is created. This dialog box contains input fields that takes input such as subject and message of the topic and records them under a shared preference key (e.g. frompostdet1 as one of the keys). The posts are associated with the user with unique IDs to separate the posted data. Using all the data gathered from the post, the post is later assigned to the text element under the discussions. Similar functions are implemented in the topic creation section for every privileged users



```
case R.id.newtopicB:

    AlertDialog.Builder mBuilder = new AlertDialog.Builder( context: this);
    View mView = getLayoutInflater().inflate(R.layout.message,  root: null);
    final EditText msubject = (EditText) mView.findViewById(R.id.subjecttxt);
    final EditText messageT = (EditText) mView.findViewById(R.id.messagetxt);
    Button bpost = (Button) mView.findViewById(R.id.postB);

    mBuilder.setView(mView);
    final AlertDialog dialog = mBuilder.create();
    dialog.show();
    bpost.setOnClickListener((v) → {
            if(loggedinuser.equals("tom")) {
                if (!msubject.getText().toString().isEmpty() && !messageT.getText().toString().isEmpty()) {
                    Toast.makeText( context: Forum.this,  text: "Sucessefuly Posted Message: " + msubject.getText().toString(), Toast.LENGTH_SHORT).show();

                    myPrefs = getSharedPreferences( name: "MyPref", Context.MODE_PRIVATE);
                    myeditor = myPrefs.edit();
                    String getname = myPrefs.getString( key: "nameStr",  defValue: "0");
                    // String postid = myPrefs.getString(getname, "0");
                    myeditor.putString("from1postdet", "From: " + loggedinuser + "\nSubject: " +msubject.getText().toString() + "\n Message: " + "\n"+messageT.getText()
                    myeditor.commit();
                    String getpost1 = myPrefs.getString( key: "from1postdet",  defValue: "0");
                    postid1.setText(getpost1);
```

Forum.java

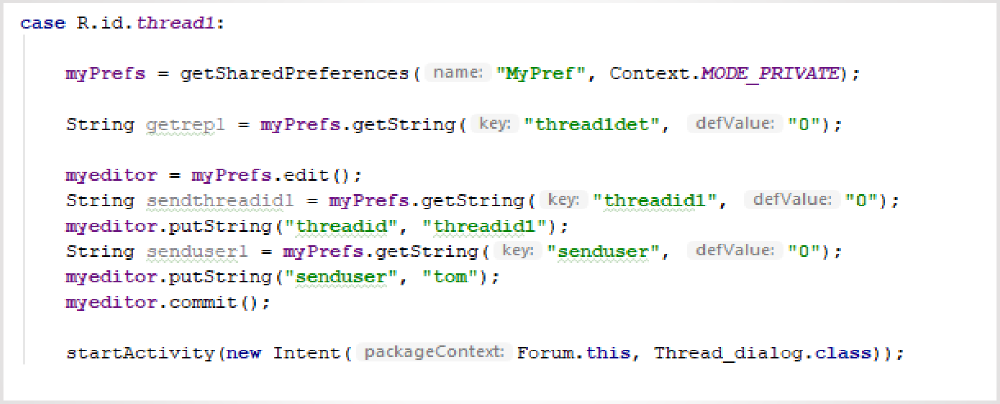
When a user decides to reply to another user's topics, the reply button listener methods process the actions that are followed. A new Dialog box is created with input field for the reply message. Shared Preference key 'threadet' is accessed and updated with the reply from the user. This key is stored with replies from different users for a particular topic/thread and later retrieved for displaying under the 'view thread' feature.



```
case R.id.reply2:

    mBuilder = new AlertDialog.Builder( context: this);
    View mView3 = getLayoutInflater().inflate(R.layout.reply,  root: null);
    final EditText reptxt2 = (EditText) mView3.findViewById(R.id.reptxt);
    Button postb2 = (Button) mView3.findViewById(R.id.postB);

    mBuilder.setView(mView3);
    final AlertDialog dialog3 = mBuilder.create();
    dialog3.show();

    postb2.setOnClickListener((v) → {
            Toast.makeText( context: Forum.this,  text: "Sucessefuly Posted Reply", Toast.LENGTH_SHORT).show();

            myPrefs = getSharedPreferences( name: "MyPref", Context.MODE_PRIVATE);
            myeditor = myPrefs.edit();
            String getreply2 = myPrefs.getString( key: "thread2det",  defValue: "");
            String appendedreply2 = getreply2+"\n\n"+"\n From: " + loggedinuser + "\n Message: " +"\n"+ reptxt2.getText().toString();
            myeditor.putString("thread2det", appendedreply2);
            myeditor.commit();
```

Forum.java

When the user accesses the thread view function, the related button listener actions are performed. The thread ID preference key is created depending on the thread/topic requested by the user. This “threadid” key value is initiated and an instance of Thread_dialog.java class (which is contains methods to draw the view thread interface) is launched.

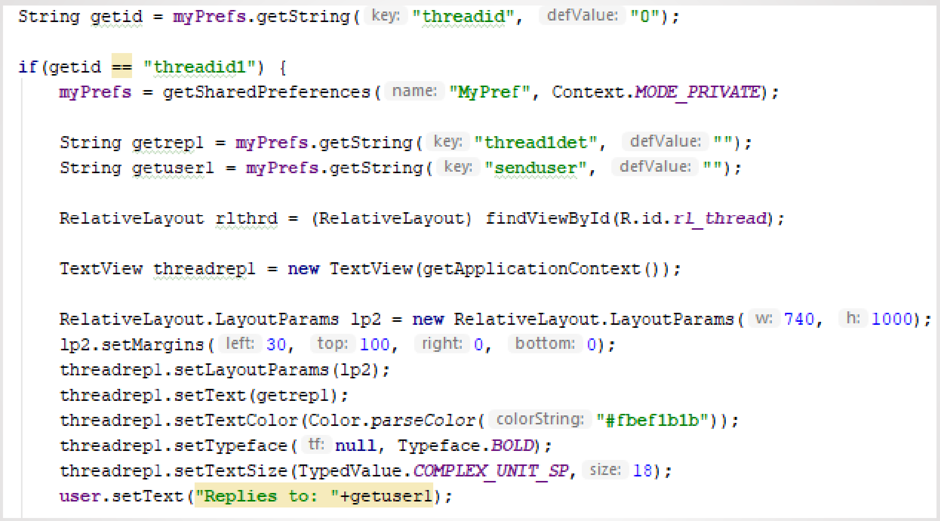


```
case R.id.thread1:

    myPrefs = getSharedPreferences( name: "MyPref", Context.MODE_PRIVATE);

    String getrepl = myPrefs.getString( key: "thread1det",  defValue: "0");

    myeditor = myPrefs.edit();
    String sendthreadid1 = myPrefs.getString( key: "threadid1",  defValue: "0");
    myeditor.putString("threadid", "threadid1");
    String senduser1 = myPrefs.getString( key: "senduser",  defValue: "0");
    myeditor.putString("senduser", "tom");
    myeditor.commit();

    startActivity(new Intent( packageContext: Forum.this, Thread_dialog.class));
```

Forum.java

The Thread_dialog.java activity which is launched by the forum class holds the functions to display the threads in a dialog interface. The unique thread identifying key “threadid” which was assigned in the forum class is retrieved. Based on an ‘if’ statement (depending on the thread ID) creates the text views with the thread data. “Layoutparams” allows for creation of text data through program instead of premade text elements. The thread ID is used to retrieve all the details of the thread and assigns them into the text elements made through layoutparams.

The properties (colour, size etc.) of the text displayed are also assigned to text elements. These functions are implemented in a similar manner for all of the thread ID’s depending on the user.



```
String getid = myPrefs.getString( key: "threadid",  defValue: "0");

if(getid == "threadid1") {
    myPrefs = getSharedPreferences( name: "MyPref", Context.MODE_PRIVATE);

    String getrepl = myPrefs.getString( key: "thread1det",  defValue: "");
    String getuser1 = myPrefs.getString( key: "senduser",  defValue: "");

    RelativeLayout rlthrd = (RelativeLayout) findViewById(R.id.rl_thread);

    TextView threadrepl = new TextView(getApplicationContext());

    RelativeLayout.LayoutParams lp2 = new RelativeLayout.LayoutParams( w: 740,  h: 1000);
    lp2.setMargins( left: 30,  top: 100,  right: 0,  bottom: 0);
    threadrepl.setLayoutParams(lp2);
    threadrepl.setText(getrepl);
    threadrepl.setTextColor(Color.parseColor( colorString: "#fbef1b1b"));
    threadrepl.setTypeface( tf: null, Typeface.BOLD);
    threadrepl.setTextSize(TypedValue.COMPLEX_UNIT_SP,  size: 18);
    user.setText("Replies to: "+getuser1);
```

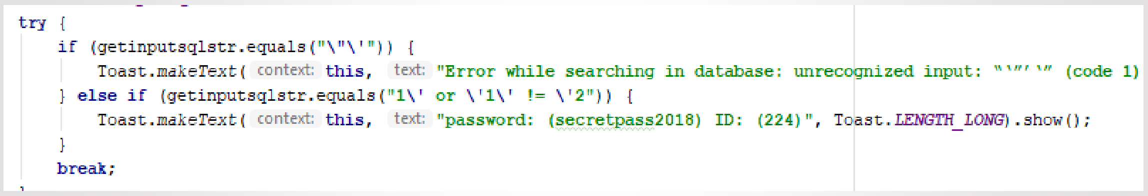
Thread_dialog.java

Guide

The guide section of the application contains the guides/learning modules for different types of vulnerabilities and unsafe code practices.

The LearnLogging.java class provides examples of logging methods and uses in application development. The class contains text and images to educate users on insecure data storage.

The LearnSQL.java class explains the implementation of databases in android applications. It focuses in providing users with examples of bad validation in functions that use databases. It contains an interactive section, where the users can test for SQL injections. If the user enters a proper SQL injection, the sample database is revealed to them through an alert message.



```
try {
    if (getinputsqlstr.equals("\"\'")) {
        Toast.makeText( context: this,  text: "Error while searching in database: unrecognized input: "\"'\" (code 1)
    } else if (getinputsqlstr.equals("1\' or \'1\' != \'2")) {
        Toast.makeText( context: this,  text: "password: (secretpass2018) ID: (224)", Toast.LENGTH_LONG).show();
    }
    break;
```

LearnSQL.java

5.3 Technologies Utilized

The main applications and technologies implemented in the development of the project are:

- Android studio 3.1.2 - IDE
- Android Phone Emulator –(AVD) Addon for Android Studio
- JAVA – Version 8
- SQLite 3.7.4
- OneDrive

6 Testing and Evaluation

6.1 Functional Testing

VulnApp

Requirements for the application:

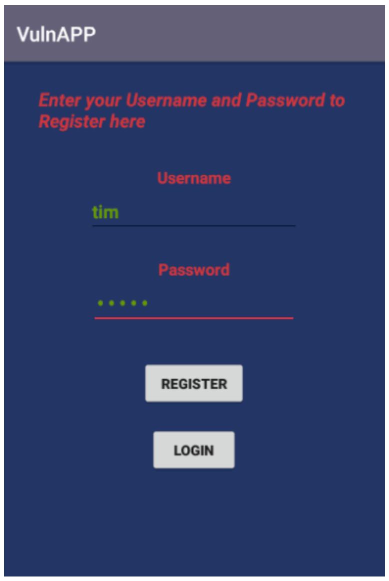
- Android build API versions 15+(Ice cream Sandwich & above)
- Internet Connection

The main tasks that are tested in the section are:

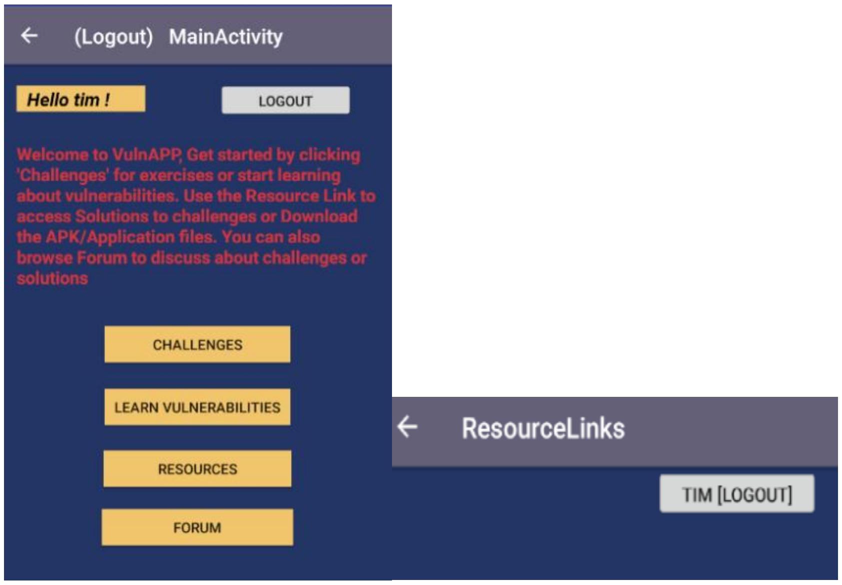
- App launching and displaying without any errors.
- Register/Login is displayed at launch.
- Logged in user status is reflected throughout the application.
- Challenge scenarios working as intended.
- Scores being added and tracked for all users.
- Forum functions working as designed, with topics, threads and replies displaying as intended
- Normal user restriction functions.

App launching and displaying without any errors, Register/Login is displayed at launch and User status is displayed on different sections of the app.

The application launches the register/login interface first as designed.



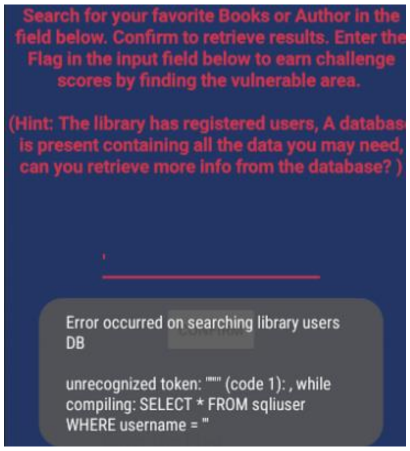
The Main activity class is working as intended displaying all the sections of the application without issues. The user login status is also tracked across sections.



Welcome to VulnAPP, Get started by clicking 'Challenges' for exercises or start learning about vulnerabilities. Use the Resource Link to access Solutions to challenges or Download the APK/Application files. You can also browse Forum to discuss about challenges or solutions

Challenge scenarios working as intended

Challenges are performing in a proper manner. The correct and incorrect inputs are treated differently as programmed.

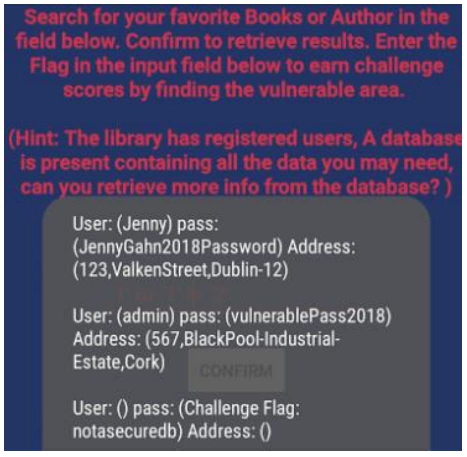


Search for your favorite Books or Author in the field below. Confirm to retrieve results. Enter the Flag in the input field below to earn challenge scores by finding the vulnerable area.

(Hint: The library has registered users, A database is present containing all the data you may need, can you retrieve more info from the database?)

Error occurred on searching library users DB

unrecognized token: "'" (code 1): , while compiling: SELECT * FROM sqliuser WHERE username = ''



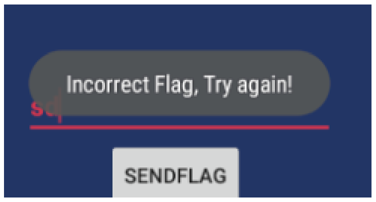
Search for your favorite Books or Author in the field below. Confirm to retrieve results. Enter the Flag in the input field below to earn challenge scores by finding the vulnerable area.

(Hint: The library has registered users, A database is present containing all the data you may need, can you retrieve more info from the database?)

User: (Jenny) pass: (JennyGahn2018Password) Address: (123,ValkenStreet,Dublin-12)

User: (admin) pass: (vulnerablePass2018) Address: (567,BlackPool-Industrial-Estate,Cork)

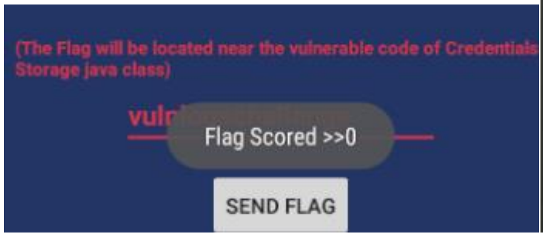
User: () pass: (Challenge Flag: notasecuredb) Address: ()



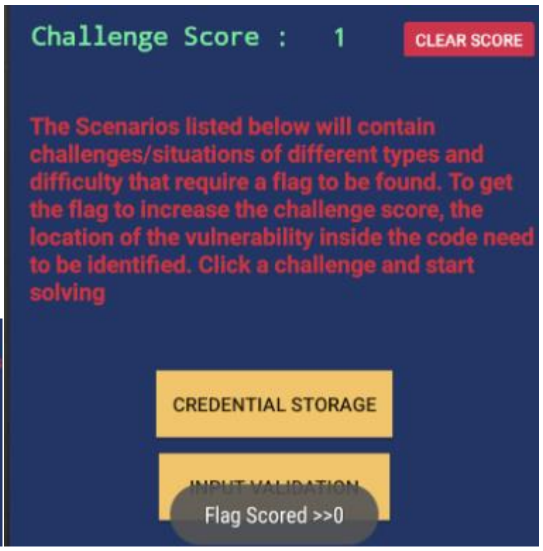
Incorrect Flag, Try again!

Scores being added and tracked for all users.

Earning scores are functioning, scores incrementing are also processed as coded.



(The Flag will be located near the vulnerable code of Credentials Storage java class)

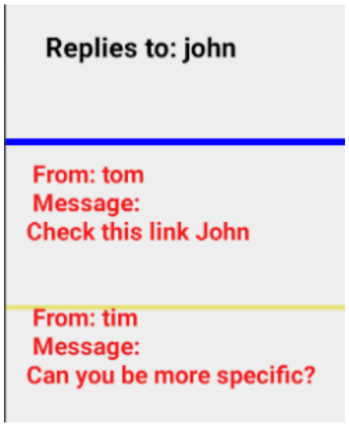


Challenge Score : 1

The Scenarios listed below will contain challenges/situations of different types and difficulty that require a flag to be found. To get the flag to increase the challenge score, the location of the vulnerability inside the code need to be identified. Click a challenge and start solving

Forum functions working as designed, with topics, threads and replies displaying as intended

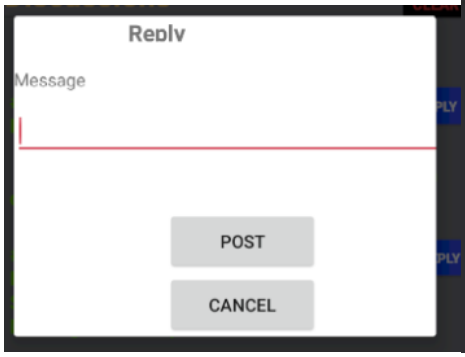
Various forum activities such as reply, viewing are in working condition.

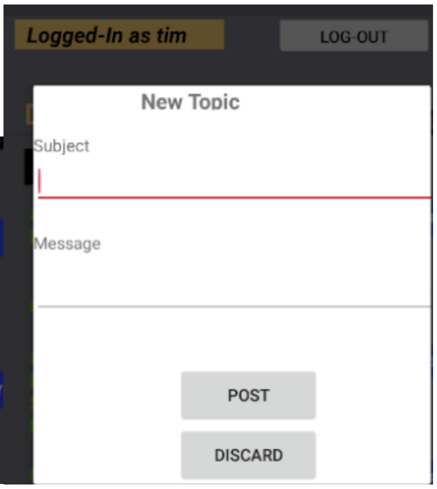


Replies to: john

From: tom
Message:
Check this link John

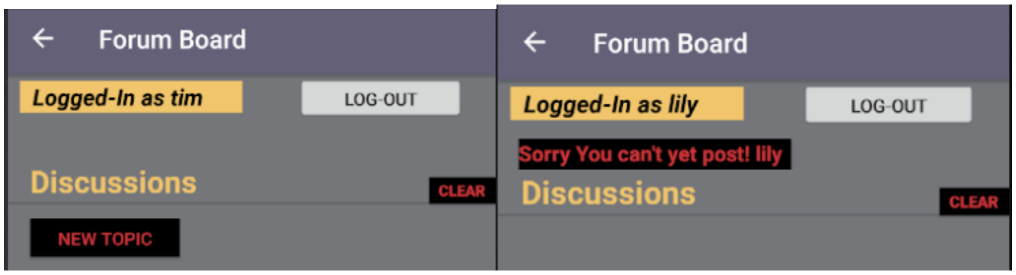
From: tim
Message:
Can you be more specific?





Normal user restriction functions working as intended

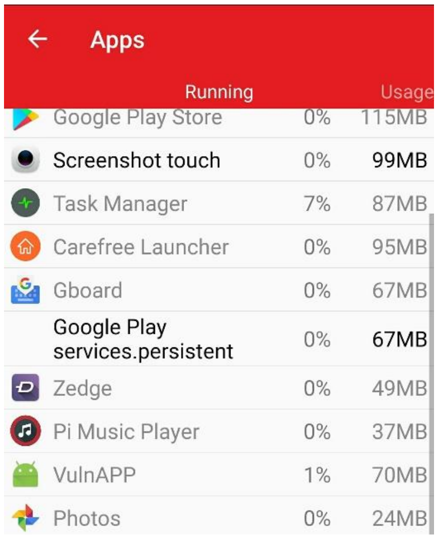
The normal users (e.g. lily) are restricted from posting new topic, while the privileged users have access to new topic function.



6.2 Conditions Testing

6.2.1 Battery Consumption and efficiency

The application is tested on BLU Vivo 6(gsmarena, 2018) which has a batter capacity of 3130mAh. The battery usage is calculated using task manager. The highest amount CPU usage is around 1% and the most memory it occupies is around 70mb across the application.



Running		Usage
Google Play Store	0%	115MB
Screenshot touch	0%	99MB
Task Manager	7%	87MB
Carefree Launcher	0%	95MB
Gboard	0%	67MB
Google Play services.persistent	0%	67MB
Zedge	0%	49MB
Pi Music Player	0%	37MB
VulnAPP	1%	70MB
Photos	0%	24MB

6.3 User Trials

The application is introduced to the target audience (beginner android user/developer) as a part of the user trials. Three users are tested with various challenge scenarios.

Test Set presented to the Introduced are:

Vulnerable Storage – Log based

Vulnerable Database – SQL based

Exposed Data

6.3.1 Challenge tests

The test users are provided 25 minutes in total to complete 3 challenges. They are also given samples of background information from books and research papers to help on solving the challenges

The following information/guide on information storage leakage from the Android Hacker's Handbook (Joshua J et al., 2014) is provided to the testers.

Analysis and Prevention of Code-Injection Attacks on Android OS Research Excerpt

Information Leakage Through Logs

Android's log facility is a great source of information leaks. Through developers' gratuitous use of log methods, often for debugging purposes, applications may log anything from general diagnostic messages to login credentials or other sensitive data. Even system processes, such as the ActivityManager, log fairly verbose messages about Activity invocation. Applications bearing the READ_LOGS permission can obtain access to these log messages (by way of the logcat command).

As an example of ActivityManager’s logging verbosity, consider the following log snippet:

```
I/ActivityManager(13738): START {act=android.intent.action.VIEW
dat=http://www.wiley.com/
cmp=com.google.android.browser/com.android.browser.BrowserActivity
(has extras) u=0} from pid 11352
I/ActivityManager(13738): Start proc com.google.android.browser for
activity com.google.android.browser/com.android.browser.BrowserActivity:
pid=11433 uid=10017 gids={3003, 1015, 1028}
```

You see the stock browser being invoked, perhaps by way of the user tapping a link in an e-mail or SMS message. The details of the Intent being passed are clearly visible, and include the URL (http://www.wiley.com/) the user is visiting. Although this trivial example may not seem like a major issue, under these circumstances it presents an opportunity to garner some information about a user’s web-browsing activity.
A more cogent example of excessive logging was found in the Firefox browser for Android. Neil Bergman reported this issue on the Mozilla bug tracker in December 2012. Firefox on Android logged browsing activity, including URLs that were visited. In some cases, this included session identifiers, as Neil pointed out in his bug entry and associated output from the logcat command:

```
I/GeckoBrowserApp(17773): Favicon successfully loaded for URL =
https://mobile.walmart.com/m/pharmacy;jsessionid=83CB330691854B071CD172D41DC2C3
AB
I/GeckoBrowserApp(17773): Favicon is for current URL =
https://mobile.walmart.com/m/pharmacy;jsessionid=83CB330691854B071CD172D41DC2C3
AB
E/GeckoConsole(17773): [JavaScript Warning: "Error in parsing value for
'background'. Declaration dropped." {file:
"https://mobile.walmart.com/m/pharmacy;jsessionid=83CB330691854B071CD172D41DC2C
3AB?wicket:bookmarkablePage=:com.wm.mobile.web.rx.privacy.PrivacyPractices"
line: 0}]
```

In this case, a malicious application (with log access) could potentially harvest these session identifiers and hijack the victim’s session on the remote web application.

Android Hacker’s Handbook EXCERPT

1.2 SQLite Injection

The Android OS ships a SQLite API with its development kit. SQL injection vulnerabilities are well studied and are the single most common vulnerability in software, according to the MITRE Common Vulnerabilities List [1]. A SQLite database in Android is not protected by a permission restriction in the application manifest but does have several countermeasures against injection. The countermeasures take the form of wrapper methods to access the database [7]. These wrapper methods all have overloads that utilize prepared statements. Prepared statements are a common

```
// Construct a SQLite query string
String query =
// Match rows where KEYVALUNAME is equal to untrusteduname and
KEYVALUNAME + ‘‘ = ’’’ + untrusteduname + ‘‘’ AND ’’ +
// Rows where KEYVALPASS is equal to untrustedpass
KEYVALPASS + ‘‘ = ’’’ + untrustedpass + ‘‘’ ’’;
// Store results from the query into a cursor
Cursor cursor = sqldatabase.query(
// Pass to the query the table to search, the columns to return
TABLEVALDEMO, new String[] { KEYVALID, KEYVALUNAME, KEYVALPASS },
// and our query string. Other optional args are null.
query, null, null, null, null, null);
// If any match was found, return true, else return false
return(cursor.moveToFirst())}
```

Figure 1.2: Vulnerable Java code.

SQL injection prevention mechanism that \prepare" a SQL query before concatenating in user input [8]. The untrusted strings are then converted to equivalent values and added to the query. We call this behavior \string binding." Closed values are noncode, so this protection mechanism fully defends against injection attacks if used properly. However, applications do not automatically implement this mechanism. The onus is on the developer to make use of these overloads. Developers who are unfamiliar with the dangers of injection attacks might not know that they need to program their applications securely.

An example of such a poorly coded application might look like Figure 1.2. The program is querying a SQLite database with columns for primarykey, username and password for the exis-tence of a particular username and password in the database, attempting to authenticate some user. It constructs a query string that implements this behavior. Note that SQLite requires that string literals in queries be enclosed with single quote characters. The single quote characters to enclose untrusteduname and untrustedpass are included in the application code, before and after concatenating in the string variables. The query string is then used as an one of the arguments to Android's SQLite API method .query(). The code also passes to the .query() method the

name of the table to be queried and which columns of information to return in the results. Other arguments, including how to sort the results and how to group the results are left as null. The query method returns an instance of the Cursor class, a datatype that contains the results of the query. The code should then return true if it has found a match to the username/password com-bination in the database and false otherwise. The .moveToFirst() method of the Cursor class implements exactly that behavior, returning false if the Cursor is empty, and true otherwise. This is a standard example of how a SQLite database might be queried in an Android application. If the source of untrusteduname and untrustedpass are controlled by an attacker, an injection attack could be executed on this application code.

Consider the following example. Suppose that our example database from Figure 1.2 contains only 1 entry|the entry for the administrator account. The username for the administrator account is \admin" and the password is \adminpass". According to our speci cations, the code in Fig-ure 1.2 should only return true if untrusteduname and untrustedpass each match the credentials on some row in the database. A malicious attacker attempts to authenticate to the application with username ``admin" and password `` ' OR`1'=`1". This results in the query String query = KEYVALUNAME + `` = ` " + admin + `` ' AND " + KEYVALPASS + `` = ` " + 'OR`1'=`1 + ``' ".

Our example database should reject this authentication attempt. The values passed into the .authenticate() method by the attacker do not match the values for the user account stored in the database.

This application code does not reject the authentication attempt as desired. The username is matched as normal but the password provided is not tokenized as a string literal. Instead, the rst single quote in the untrusted input closes the string literal that should have contained the password to match against the password eld in the database. The untrusted input then injects an extra OR token and an equality expression `1'=`1. Recall that SQLite also requires that literals be enclosed with single quotes. The single quote to the right hand side of the nal number `1' is left absent in the user provided string because the trusted query string has one already|the single quote that would have closed the password string literal. This injection results in the SQL query matching all rows where the column KEYVALUNAME is equal to \admin" and rows where the column KEYVALPASS is equal to \ " or where \1 = 1". The equation \1 = 1" is, of course, a tautology, resulting in

```
// Construct the query string marking user input locations with ?'s.
String query = KEYVALLOGIN + ‘‘ =? ’’+ ‘‘AND’’ + KEYVALPASS + ‘‘=?’’;
// Pass to the query method the table to search,
Cursor cursor = db.query(TABLEVALDEMO,
// the columns to return, the query string
new String[] { KEYVALID, KEYVALLOGIN, KEYVALPASS }, query,
// and String array containing untrusted input to be bound as literals
new String[] { untrusteduname, untrustedpass}, null, null, null, null);
```

Figure 1.3: Code for an application that uses Android's SQL injection security mechanism

the query matching every row in the database. A row is returned, so the conditional at the end of the code returns true and the malicious user successfully authenticates to the database, despite not providing a valid username/password combination.

The injection attack described above was made possible by a poorly programmed application. The code of an equivalent application that takes advantage of

Android's protection mechanisms would look like the code in Figure 1.3. This application code makes use of a prepared statement. The programmer can use a prepared statement by marking areas where untrusted input will be used in the query with `?' characters. The example code does this for both the untrusuteduname and untrustedpass strings. Then, when the SQLite API method is called with our query string, the fourth argument is used to pass in a String array containing the untrusted strings. It is then the task of the API method to insert the untrusted strings into the query and ensure that they are bound as literals, meaning that the untrusted strings will be tokenized as literals only. The new secure application, instead of searching for rows for which either the password value is the empty string or the equation \1 = 1" is true, will now search for rows for which the password value is '   OR `1'=`1. Since there is no row with that password in our example database, the query returns no rows, the conditional statement correctly returns false, and the injection attack is prevented. This mechanism for preventing SQL injection attacks is present in the o -the-shelf Android OS but is not automatic for all databases. The onus is on the programmer to understand why these overloads are important and to use them properly in their code to secure their application against injection attacks.

So far as we have been able to determine, there is no reason why an application developer would not want to use the provided SQLite security mechanisms, if s/he were aware of them. If there is no penalty for using the protection mechanism, why is SQL injection still a problem? The documenta-tion for the SQLiteDatabase class describes the argument used to implement prepared statements as \You may include ?'s in selection, which will be replaced by the values from selectionArgs, in order that they appear in the selection. The values will be bound as Strings. [7]" This explanation makes no mention of what string binding is or why it is important to protect the application from injection attacks. The lack of detail might lead a developer unaware of the dangers of injection attacks to forego using the security mechanism, leaving their code vulnerable to injection attacks. In addition, prepared statements have been a feature of the Android OS since API level one|the initial release of Android in 2008. Despite the existence of these prepared statements, SQL injection continues to be a problem. Clearly, many developers are not understanding or simply neglecting to use these security features.

Following table displays the summary of the test results

User	Challenge(s) provided	Completion	Time provided	Time taken
Beginner1	Vulnerable Storage, Vulnerable Database, Exposed Data	3 challenges	30	13
Beginner2	Vulnerable Storage, Vulnerable Database, Exposed Data	3 challenges	30	25
Beginner3	Vulnerable Storage, Vulnerable Database, Exposed Data	2 challenges	30	30

Feedback and Conclusion

Beginnner1, the user who completed all challenges deemed the challenges were easier than expected. They displayed a desire to do more challenges that are harder. They also believed that Guide sections had too much 'Hand holding' making things easier.

Beginnner2 who also completed challenges deemed the challenges were around easy to medium in difficulty. They also mentioned that Guide section on the application were easier to read through as they were short and concise to the point as opposed to excerpts from book and research paper. They also felt that the interactivity of the Guide section helped them in focusing on specific developmental platform's (Android Studio) functionalities and how they illustrated the specifics of each area of vulnerability and data insecurity.

Beginnner3 who completed 2 of the 3 challenges deemed the challenges were simple for the most part. They had issues with SQL injection where they couldn't figure out the correct injection queries to extract the info from database. They also felt that Guide section needed more examples in some cases to further explain risk areas.

6.3.2 Usability/Features Test

The users were also requested to provide feedback on usability and the applications features. They were provided, 5 minutes with the application. They were also requested to report on any errors or inconveniences.

Feedback and Conclusion

After testing the app for around 5 minutes, the users provided their feedback and requests for features. Overall the users were satisfied with the UI design and usability of the application. The experience of using the app felt intuitive. No errors were spotted across the application. They also expressed that more resource references need to be added for further information/learning. They liked the offline aspect of the application though they would have like to solve more challenge based on only utilizing the app. They expressed that the flag system for challenges, increased interest in doing the challenges as they liked the hunting aspect. They note that forum is a good feature but want more functionalities and more users added to them and maybe hosting the forum online.

Requested Features

- More Challenges and Guides in general.
- More Challenges involving just mobile app.
- More Resources/References for more information on Vulnerabilities.
- Profile for accounts which records statistics.
- Features for forum such as more topic/reply functions, online implementation

7 Conclusion and Further Work

In conclusion, the project has achieved most of its core goals that were created at the planning stage. The goal of providing a base for learning that focuses on educating android users on importance of safe programming is fulfilled. The native risks involved in various technologies such as database within android is illustrated under different interactive sections (guide, challenges).

Through the feedback acquired it was easier to gauge the success of the applications intentions. In light of the feed other features that weren't added are planned for future updates. User profile being one of the requests from the user trails would be added to the application. The forum functionality to be updated with improvements such as bigger forum, migration to online hosted platform etc. In regards to challenges adding more guides to provide more information on android security. More Challenge scenarios on various difficulties that are optimized for effective learning. Overall application smoothening would be carried out.

8 Project Diary

The project diary describes significant development events over the timeline of the project.

- **February** (First half) :
  - The Project Ideas are researched in the area of Mobile Security. Project Idea is finalized on the area of Android Security.
- **February** (Second half) :
  - Researching the project background, assessing the scope of the project and main functionalities.
  - Coding the main interface along with two scenarios based on android vulnerabilities.
  - A Flag system is added to challenges.
- **March** :
  - Score System is added to the challenges
  - User register/login system is added
- **April**-**May**:
  - Forum functionalities are designed and coded.
  - Guide and resource link sections are added to the application.
  - More vulnerability challenges are added.
  - Coding the main interface along with two scenarios based on android vulnerabilities.

Appendix A- Bibliography

Blog.lookout. (2014) Heartbleed Detector. [online] Available at: https://blog.lookout.com/heartbleed-detector [Accessed 16 February 2018].

OWASP. (2016) Top Ten Mobile Vulnerabilities [online] Available at: https://www.owasp.org/index.php/Mobile_Top_10_2016-Top_10 [Accessed 17 February 2018].

Norton. (2015) StageFright Bug. [online] Available at: https://us.norton.com/internetsecurity-emerging-threats-newly-discovered-bug-stagefright-can-secretly-sneak-onto-android-phones-via-mms-text-messages.html [Accessed 17 February 2018].

CVE. (2018) Android: Vulnerability Statistics. [online] Available at: https://www.cvedetails.com/product/19997/Google-Android.html?vendor_id=1224 [Accessed 19 February 2018].

Developers, Android. (2018) Security Tips. [online] Available at: https://developer.android.com/training/articles/security-tips.html [Accessed 20 February 2018].

Oracle. (2018) Prepared Statements. [online] Available at: https://docs.oracle.com/javase/tutorial/jdbc/basics/prepared.html [Accessed 20 February 2018].

OWASP. (2015) OWASP's SeraphDroid Project. [online] Available at: https://www.owasp.org/index.php/OWASP_SeraphimDroid_Project. [Accessed 20 February 2018].

Sololearn. (2017) [online] SoloLearn: Learn to Code for Free Available at: https://play.google.com/store/apps/details?id=com.sololearn [Accessed 15 March 2018].

DVIA. (2015) Vulnerable IOS application. [online] Available at: http://damnvulnerableiosapp.com/#about [Accessed 20 February 2018].

Smartsheet. (2017) Development Models Comparison. [online] Available at: https://www.smartsheet.com/agile-vs-scrum-vs-waterfall-vs-kanban [Accessed 27 February 2018].

Looyestyn J, Kernot J, Boshoff K, Ryan J, Edney S, Maher C .(2017) Does gamification increase engagement with online programs? A systematic review. PLoS ONE 12(3): e0173403. [online] Available at: https://doi.org/10.1371/journal.pone.0173403 [Accessed 20 February 2018].

María-Blanca Ibáñez, Ángela Di-Serio, Carlos Delgado-Kloos. (2014) Gamification for Engaging Computer Science Students in Learning Activities: A Case Study https://www.researchgate.net/publication/265789392_Gamification_for_Engaging_Computer_Science_Students_in_Learning_Activities_A_Case_Study [online] Available at: https://doi.org/10.1371/journal.pone.0173403 [Accessed 20 February 2018].

Statista. (2018) Worldwide mobile app revenues in 2015, 2016 and 2020 (in billion U.S. dollars) [online] Available at: https://www.statista.com/statistics/269025/worldwide-mobile-app-revenue-forecast/ [Accessed 20 March 2018].

Draw.io. (2018) Technical Diagram software [online] Available at: https://www.draw.io/

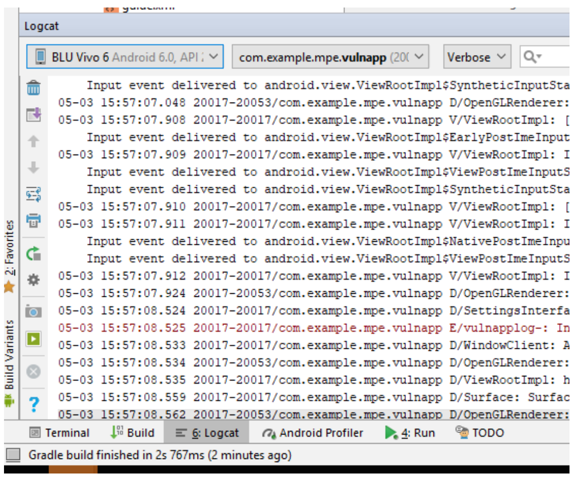
Solutions. (2018) Application Challenge Solutions [online] Available at: https://studentitb-my.sharepoint.com/:w:/g/personal/b00077857_student_itb_ie/ERVx-5opj1VJoFQMwz9fD_MB_RsqVXRpWYz96Ajn_pmRkQ

gsmarena. (2018) gsmarena- mobile phone specification lookup [online] Available at: https://www.gsmarena.com/blu_vivo_6-8426.php

Joshua J. Drake, Zach Lanier, Collin Mulliner, Pau Oliva Fora, Stephen A. Ridley,Georg Wicherski. (2014). Android Hacker's Handbook ,pp.87-88 [online] Available at: https://books.google.ie/books?id=2qo6AwAAQBAJ&pg=PR26&lpg=PR26&dq=android+vulnerabilities+book&source=bl&ots=0mvPOfaaiS&sig=T9UJuIjWCwFTyV7y8dVRWDV3IOE&hl=en&sa=X&ved=0ahUKEwiitI6H9LbaAhXrA8AKHey0DqAQ6AEIcTAI#v=onepage&q=87&f=false [Accessed May 2018].

Grant Joseph Smith. (2014) Analysis and Prevention of Code-Injection Attacks on Android OS, pp.3- 5 [online] Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.1009.3941&rep=rep1&type=pdf [Accessed May 2018].

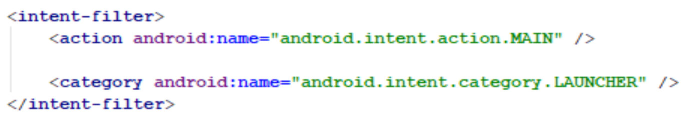
logcat.png



```
     Input event delivered to android.view.ViewRootImpl$SyntheticInputSta
05-03 15:57:07.048 20017-20053/com.example.mpe.vulnapp D/OpenGLRenderer:
05-03 15:57:07.908 20017-20017/com.example.mpe.vulnapp V/ViewRootImpl: [
     Input event delivered to android.view.ViewRootImpl$EarlyPostImeInput
05-03 15:57:07.909 20017-20017/com.example.mpe.vulnapp V/ViewRootImpl: I
     Input event delivered to android.view.ViewRootImpl$ViewPostImeInputS
     Input event delivered to android.view.ViewRootImpl$SyntheticInputSta
05-03 15:57:07.910 20017-20017/com.example.mpe.vulnapp V/ViewRootImpl: [
05-03 15:57:07.911 20017-20017/com.example.mpe.vulnapp V/ViewRootImpl: I
     Input event delivered to android.view.ViewRootImpl$NativePostImeInpu
     Input event delivered to android.view.ViewRootImpl$ViewPostImeInputS
05-03 15:57:07.912 20017-20017/com.example.mpe.vulnapp V/ViewRootImpl: I
05-03 15:57:07.924 20017-20053/com.example.mpe.vulnapp D/OpenGLRenderer:
05-03 15:57:08.524 20017-20017/com.example.mpe.vulnapp D/SettingsInterfa
05-03 15:57:08.525 20017-20017/com.example.mpe.vulnapp E/vulnapplog-: In
05-03 15:57:08.533 20017-20017/com.example.mpe.vulnapp D/WindowClient: A
05-03 15:57:08.534 20017-20053/com.example.mpe.vulnapp D/OpenGLRenderer:
05-03 15:57:08.535 20017-20017/com.example.mpe.vulnapp D/ViewRootImpl: h
05-03 15:57:08.559 20017-20017/com.example.mpe.vulnapp D/Surface: Surfac
05-03 15:57:08.562 20017-20053/com.example.mpe.vulnapp D/OpenGLRenderer:
```

Gradle build finished in 2s 767ms (2 minutes ago)

intent.png



```
<intent-filter>
    <action android:name="android.intent.action.MAIN" />

    <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

Solutions

Setup and Installation

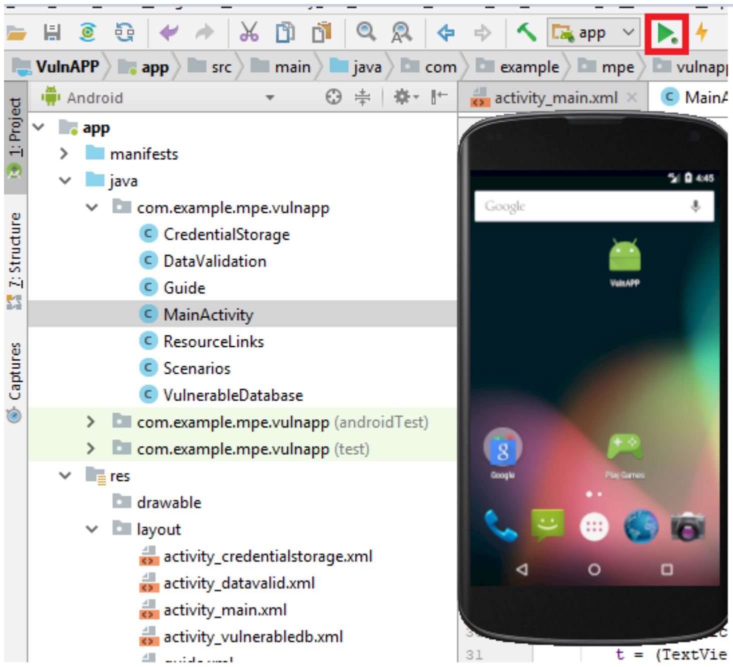
Tools Required:

Android Studio (+Android SDK+ Android OS image for HAMX emulation), Project Application files.

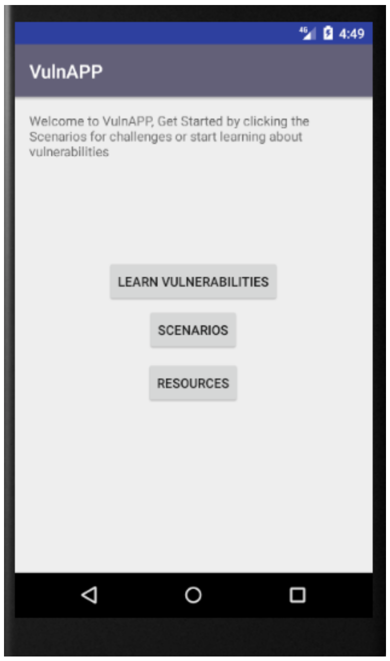
Walkthrough:

1) Install Android Studio and its SDK. Create an Android device with OS (above 4.0 Ice Cream Sandwich) image to the built in Emulator.

2) Open the Project folder in Android Studio. Click on the highlighted button to start the emulator which will automatically install the application inside the phone.



3) The Application should open as soon as the APK file is installed.



Welcome to VulnAPP, Get Started by clicking the Scenarios for challenges or start learning about vulnerabilities

Challenges

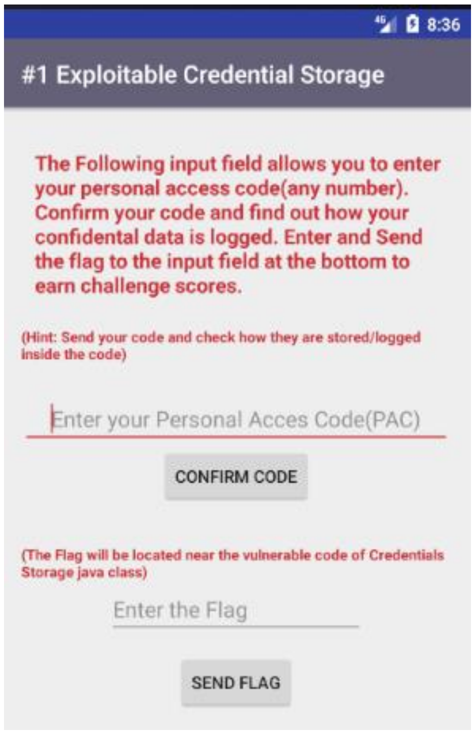
Challenge 1: Insecure Logging

Tools Required:

Android Studio (+Android SDK+ Android OS image for HAMX emulation), Project Application files.

Walkthrough:

1) Open the Project folder in Android Studio and Run the Application.


8:36
#1 Exploitable Credential Storage

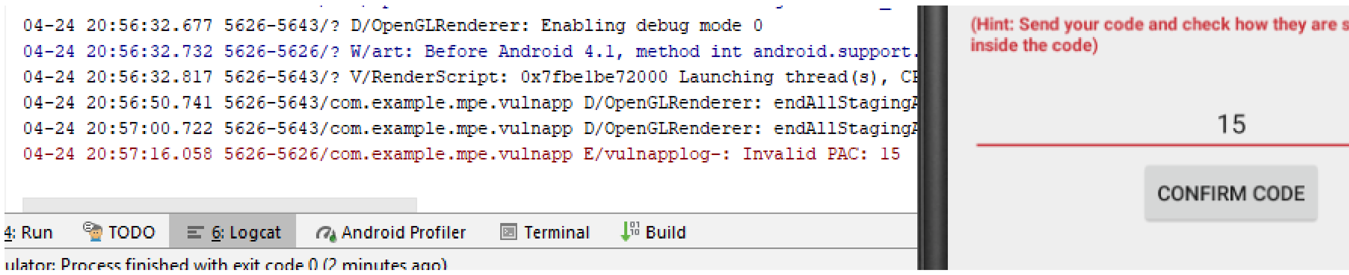
The Following input field allows you to enter your personal access code(any number). Confirm your code and find out how your confidental data is logged. Enter and Send the flag to the input field at the bottom to earn challenge scores.

(Hint: Send your code and check how they are stored/logged inside the code)

Enter your Personal Acces Code(PAC)
CONFIRM CODE

(The Flag will be located near the vulnerable code of Credentials Storage java class)

2) Navigate to the Log cat function at the bottom of the Android Studio UI while the app is running. Input any number into the first field and confirm the code.

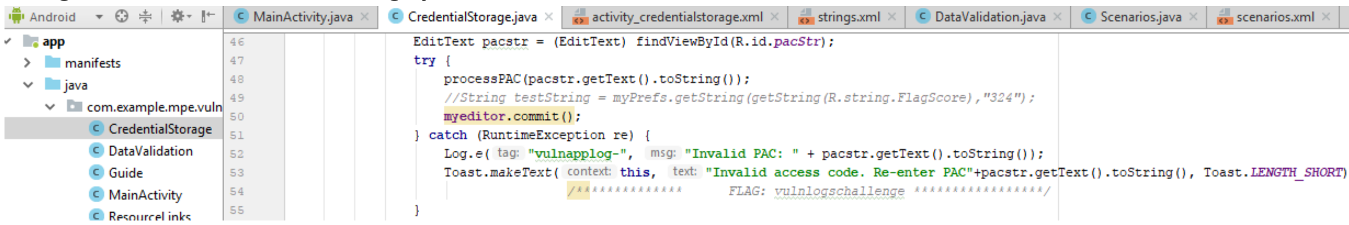


```
04-24 20:56:32.677 5626-5643/? D/OpenGLRenderer: Enabling debug mode 0
04-24 20:56:32.732 5626-5626/? W/art: Before Android 4.1, method int android.support.
04-24 20:56:32.817 5626-5643/? V/RenderScript: 0x7fbe1be72000 Launching thread(s), CP
04-24 20:56:50.741 5626-5643/com.example.mpe.vulnapp D/OpenGLRenderer: endAllStagingA
04-24 20:57:00.722 5626-5643/com.example.mpe.vulnapp D/OpenGLRenderer: endAllStagingA
04-24 20:57:16.058 5626-5626/com.example.mpe.vulnapp E/vulnapplog-: Invalid PAC: 15
```

4: Run
TODO
6: Logcat
Android Profiler
Terminal
Build
ulator: Process finished with exit code 0 (2 minutes ago)
(Hint: Send your code and check how they are s
inside the code)
15
CONFIRM CODE

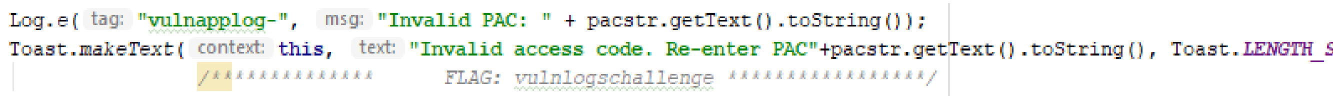
The input you’ve entered can be seen as its being logged.

3) Navigate to the Credential Storage java file and to the line 52



```
46            EditText pacstr = (EditText) findViewById(R.id.pacStr);
47            try {
48                processPAC(pacstr.getText().toString());
49                //String testString = myPrefs.getString(getString(R.string.FlagScore),"324");
50                myeditor.commit();
51            } catch (RuntimeException re) {
52                Log.e( tag: "vulnapplog-",  msg: "Invalid PAC: " + pacstr.getText().toString());
53                Toast.makeText( context: this,  text: "Invalid access code. Re-enter PAC"+pacstr.getText().toString(), Toast.LENGTH_SHORT)
54                                /**************       FLAG: vulnlogschallenge ****************/
55            }
```

4) The line 51, 52 contains the code to store and log wrong attempts of inputs on the PAC field. An example of exploitable record especially if the input is a confidential credential.



```
Log.e( tag: "vulnapplog-",  msg: "Invalid PAC: " + pacstr.getText().toString());
Toast.makeText( context: this,  text: "Invalid access code. Re-enter PAC"+pacstr.getText().toString(), Toast.LENGTH_S
                /**************       FLAG: vulnlogschallenge ****************/
```

5) The Flag can also be found in the lines below the vulnerable code.

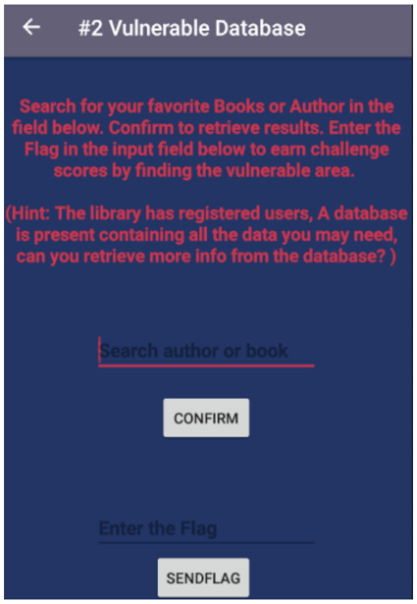
Challenge 2: Vulnerable Database

Tools Required:

Android Studio (+Android SDK+ Android OS image for HAMX emulation), Project Application files.

Walkthrough:

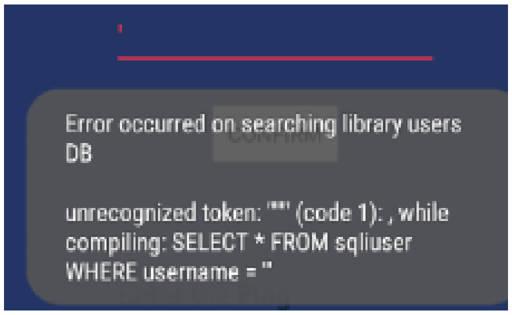
1. Launch the Vulnerable Database challenge. The challenge requires you to get information from the database(users)



Search for your favorite Books or Author in the field below. Confirm to retrieve results. Enter the Flag in the input field below to earn challenge scores by finding the vulnerable area.

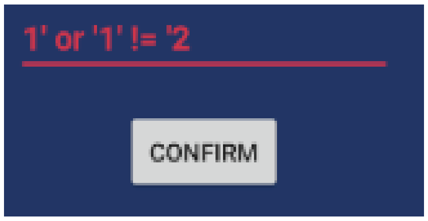
(Hint: The library has registered users, A database is present containing all the data you may need, can you retrieve more info from the database?)

2. Enter single quotes to into the search fields to retrieve the any possible info about the database.



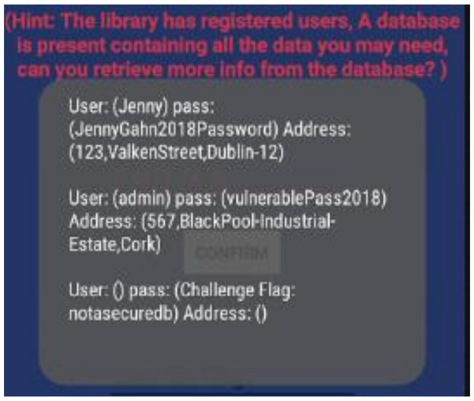
```
'

Error occurred on searching library users
DB

unrecognized token: "''" (code 1): , while
compiling: SELECT * FROM sqliuser
WHERE username = ''
```

3. Enter 1’ or ‘1’ = ‘2 into the fields which will reveal an alert message, the Flag will be on one of the lines inside the alert message



```
1' or '1' != '2
```

4. The flag is found on the last line of the retrieved database result


(Hint: The library has registered users, A database is present containing all the data you may need, can you retrieve more info from the database?)

User: (Jenny) pass: (JennyGahn2018Password) Address: (123,ValkenStreet,Dublin-12)

User: (admin) pass: (vulnerablePass2018) Address: (567,BlackPool-Industrial-Estate,Cork)

User: () pass: (Challenge Flag: notasecuredb) Address: ()

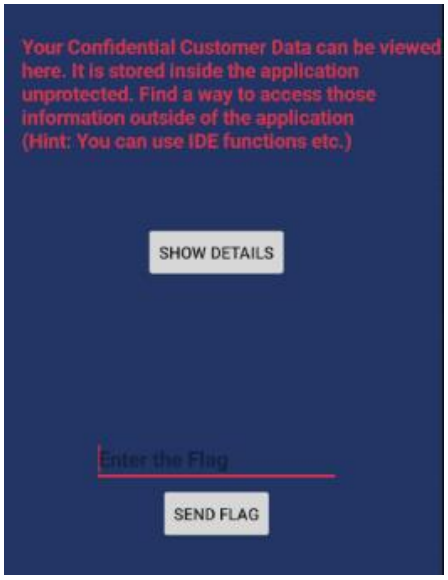
Challenge 3: Vulnerable Access

Tools Required:

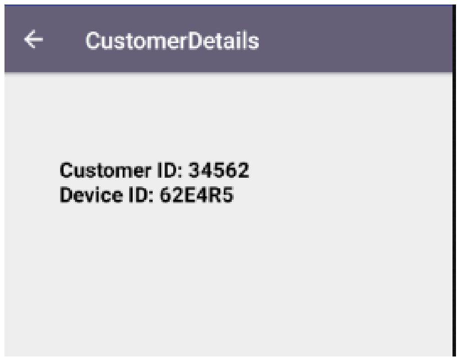
Android Studio (+Android SDK+ Android OS image for HAMX emulation), Project Application files. Environment PATH set to adb shell.

Walkthrough:

1. Launch the Vulnerable Access challenge

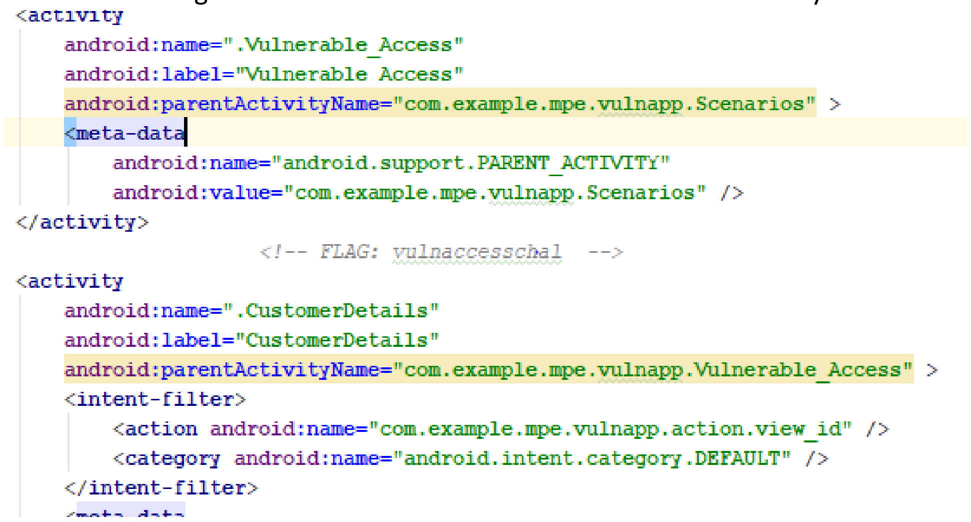

Your Confidential Customer Data can be viewed here. It is stored inside the application unprotected. Find a way to access those information outside of the application (Hint: You can use IDE functions etc.)

2. Click on the View details to check for the details



Customer ID: 34562
Device ID: 62E4R5

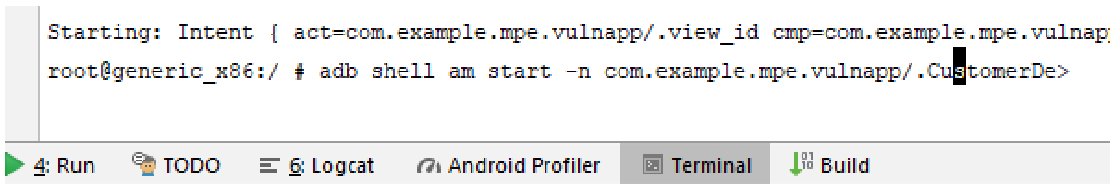
3. Now locate the following lines in the android manifest file where this activity is initiated.



```
<activity
    android:name=".Vulnerable_Access"
    android:label="Vulnerable Access"
    android:parentActivityName="com.example.mpe.vulnapp.Scenarios" >
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value="com.example.mpe.vulnapp.Scenarios" />
</activity>
                    <!-- FLAG: vulnaccesschal  -->
<activity
    android:name=".CustomerDetails"
    android:label="CustomerDetails"
    android:parentActivityName="com.example.mpe.vulnapp.Vulnerable_Access" >
    <intent-filter>
        <action android:name="com.example.mpe.vulnapp.action.view_id" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
```

4. The Flag is located close to the intent filter. Note the Intent filter action(view_id) associated with the customer details initiation.

5. Now try accessing the customer details from the terminal in android studio



```
Starting: Intent { act=com.example.mpe.vulnapp/.view_id cmp=com.example.mpe.vulnap
root@generic_x86:/ # adb shell am start -n com.example.mpe.vulnapp/.CustomerDe>
```

6. Type the following command into the terminal -
   adb shell am start -n com.example.mpe.vulnapp/.CustomerDetails -a com.example.mpe.vulnapp/.view_id

7. This command should to start the customer details activity, which gives you access to the confidential customer details screen outside of the application.

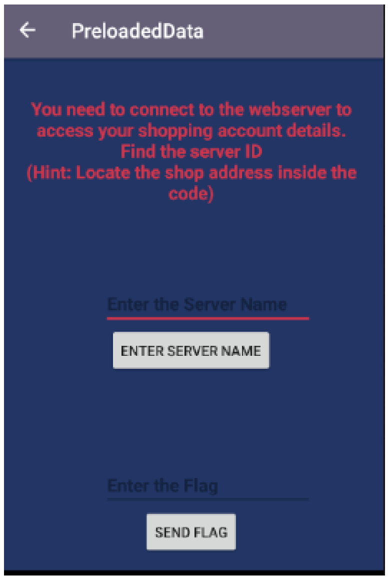
Challenge 4: Exposed Data

Tools Required:

Android Studio (+Android SDK+ Android OS image for HAMX emulation), Project Application files.

Walkthrough:

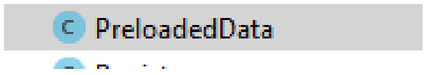
1. Launch the Exposed data challenge


PreloadedData

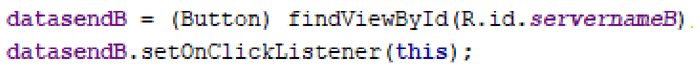
You need to connect to the webserver to access your shopping account details. Find the server ID
(Hint: Locate the shop address inside the code)

2. This challenge requires you to find the location of the server name ID

3. Go to the PreloadedData.java class file

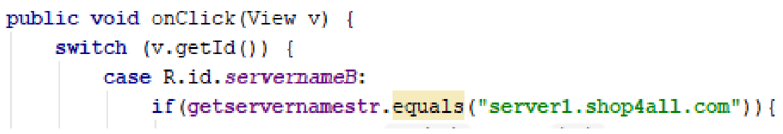


4. Search for the button actions related to the "Enter server name " from the activity.



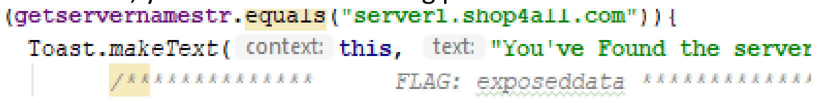
```
datasendB = (Button) findViewById(R.id.servernameB)
datasendB.setOnClickListener(this);
```

5. Under that buttons onClick method, you can now see a domain address



```
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.servernameB:
            if(getservernamestr.equals("server1.shop4all.com")){
```

6. Close to the server name, you can also see the flag present.



```
(getservernamestr.equals("server1.shop4all.com")){
 Toast.makeText( context: this,  text: "You've Found the server
       /**************          FLAG: exposeddata ************
```