Damn Vulnerable Android Application

A vulnerable application created to educate developers and security professionals alike

ZenTech

Martin Georgiev

CMP311: Professional Project Development & Delivery

BSc Ethical Hacking

2021/22

10/05/2022

Note that Information contained in this document is for educational purposes only.

Contents

2 METHOD

2.1 OVERVIEW

The development process of the Damn Vulnerable Android App followed a strict methodology due to the time constraints and the nature of the project. It had the following phases – Base application, Vulnerability One, Two and Three, respectively. Separating the development process into the aforementioned phases aided with the accurate delivery and quality of the project.

2.2 ENVIRONMENT AND LANGUAGE

DVAA was developed with the use of Expo, a React Native library. React Native is a JavaScript framework that allows prompt creation of mobile software for Android and iOS devices and is suitable for people with little to no knowledge of mobile development. Expo makes the process even easier by automatically managing parts of the project instead of the developer. Those were the biggest reasons for the choice of the team - the range of devices and the popularity among inexperienced developers. (Boughdiri, 2021) Such conditions would create the perfect environment for severe vulnerabilities.

Additionally, ZenTech used two applications for the development – Visual Studio Code (**Figure 2.2.1**) and GitHub (**Figure 2.2.2** and **Appendix D**)**.** The former, an IDE, ensured smooth and quick coding practices due to the substantial number of available plugins. The latter was used to store the code, submit any changes in real time and provide easy access to both developers and clients.

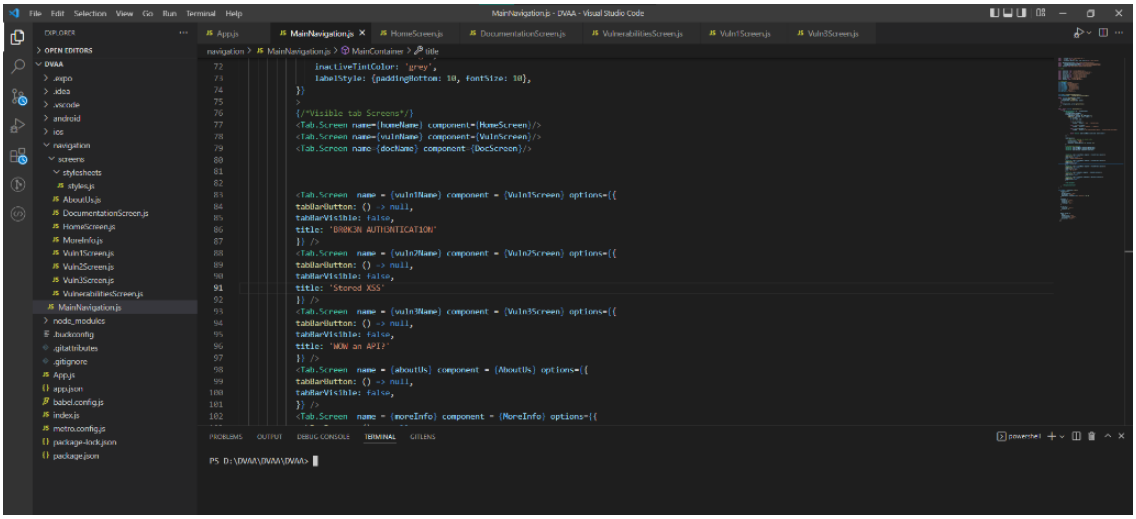

Figure 2.2.1 *- Editing the project in VS Code.*

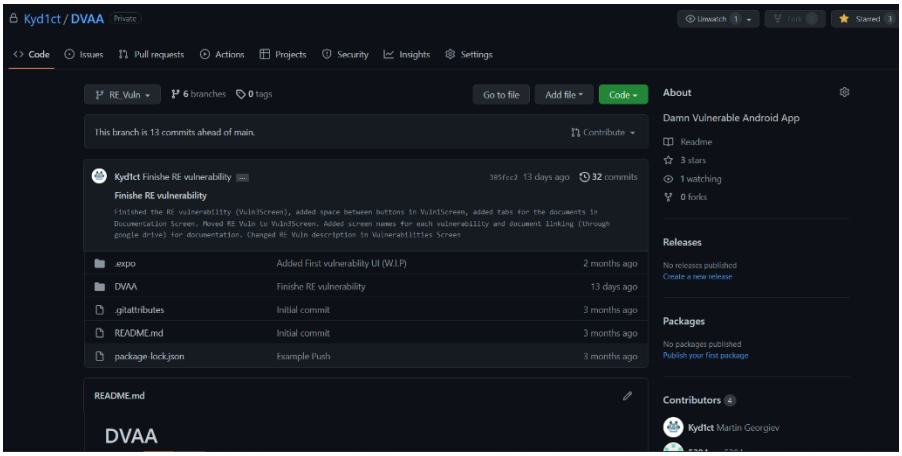

Figure 2.2.2 *- DVAA's Github page.*

Furthermore, Expo allowed the team to gather quick and efficient feedback as a simple QR Code scan would run the application on a physical device, whilst any code alterations would be applied in real time.

2.3 BASE APPLICATION

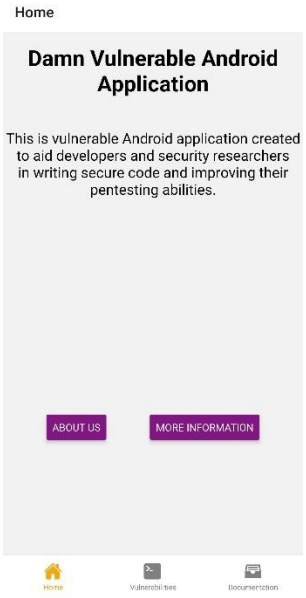

Figure 2.3.1 *- Home screen.*

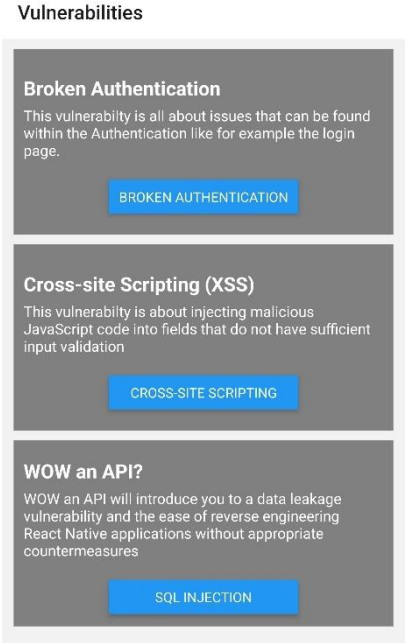

Figure 2.3.2 *- Vulnerabilities screen.*

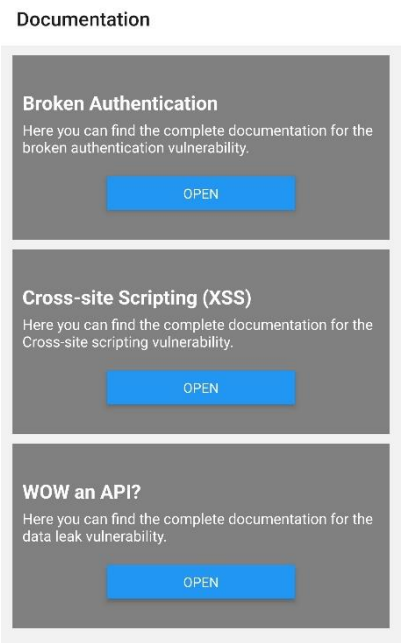

Figure 2.3.3 *- Documentation screen.*

The application was created with three main screens – Home Screen, Vulnerabilities Screen and Documentation Screen (**Figure 2.3.1**). The Home Screen held the title and brief information regarding the application and two buttons – one leading to the About Us subscreen and linking the user to the More Info subscreen. The second screen held each of the separate vulnerabilities. Every entry had a field with the title of the task, a brief description and a button leading to the appropriate subscreen (**Figure**

2.3.2 and **Appendix B**). The third screen had a similar design to the Vulnerabilities Screen, but the buttons lead to the documentation for each of the developed vulnerabilities (**Figure 2.3.3**).

Traversing the application was achieved with the use of a bottom navigation bar. The developers created a separate file, which handled the navbar and its navigation loop. (**Appendix A**) The three main screens were displayed in the bottom bar, with suitable icons for a better user experience. The file also handled the subscreens to ensure that no nested navigation loops were used. They were then hidden from the navigation bar by removing their title and covered space (**Figure 2.3.4**).

```
<Tab.Screen  name = {vuln1Name} component = {Vuln1Screen} options={{
tabBarButton: () => null,
tabBarVisible: false,
title: 'BR0K3N AUTH3NTICAT1ON'
}} />
<Tab.Screen  name = {vuln2Name} component = {Vuln2Screen} options={{
tabBarButton: () => null,
tabBarVisible: false,
title: 'Stored XSS'
}} />
<Tab.Screen  name = {vuln3Name} component = {Vuln3Screen} options={{
tabBarButton: () => null,
tabBarVisible: false,
title: 'WOW an API?'
}} />
<Tab.Screen  name = {aboutUs} component = {AboutUs} options={{
tabBarButton: () => null,
tabBarVisible: false,
}} />
<Tab.Screen  name = {moreInfo} component = {MoreInfo} options={{
tabBarButton: () => null,
tabBarVisible: false,
}} />
```

Figure 2.3.4 *- Subscreens hidden from the navigation bar.*

2.4 DATA LEAKAGE

The Data Leakage vulnerability was developed after thorough research of a React Native apk's contents. (secureITmania, 2021) The developers used a decoy initialisation function for Google's DialogFlow API for this vulnerability (**Figure 2.4.1 and Appendix C**). A genuine API key and account were not required as the DialogFlow API did not need to work for the exploitation process.

```
class DialogFlow extends Dialogflow {
constructor(props) {
    super(props);

        Dialogflow_V2.setConfiguration(
            "your-dialogflow-project@asdf-76866.iam.gserviceaccount.com",
            '-----BEGIN PRIVATE KEY-----\nMIIEvgIBADAN...1oqOawdddddiojdq9wq09wd81u1ijzcjkllasd\n-----END PRIVATE KEY-----\n',
            Dialogflow_V2.LANG_ENGLISH,
            'testv2-3b4dss',
            'FLAG{WOW_API}'
        );
    }
}
```

Figure 2.4.1 *- Dialog Flow implementation.*

The team decided to format this vulnerability as a Capture the Flag challenge due to its flag seeking nature. The fake API key and flag can be obtained through the following methodology:

- Extracting the data from the APK file
- Analysing the directories
- Analysing each file

React Native applications contain a file called app.bundle (or index.android.bundle) which is located inside the “assets” directory. The file embodied the code of the application, having all its functions displayed in clear text. This would allow an attacker to obtain sensitive data such as API keys or login details and potentially cause severe damage.

The screen for this vulnerability consists of a user input field and three buttons. The input would be used by the tester or developer using the app to check the flag they have found. The three separate buttons had the following functions (**Figure 2.4.2**):

- Submit – A button submitting the answer and giving an appropriate alert
- Hints – Displays an alert window with clues
- Walkthrough – Opens the appropriate documentation

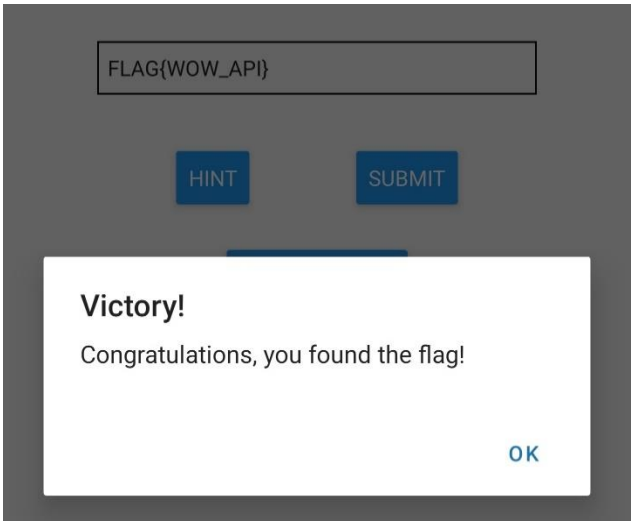

Figure 2.4.2 *- Data Leakage screen.*

2.5 BROKEN AUTHENTICATION

The developers created the broken authentication vulnerability by implementing insecure storage of passwords, detailed error messages and a lack of input sanitisation. (ReactNative, 2022) The first issue was tackled with the use of an SQLite database which stored user details such as username and password. None of the passwords were encrypted, which is a severe breach of privacy and could be disastrous for a company. The error messages provided the user with detailed information about what was incorrect – either the username or the password. The verbose errors opened possibilities for brute-force attacks. The last issue, lack of input sanitisation, was achieved by not including any string trimming scripts, allowing the users to run commands which could be paired with SQLi vulnerabilities.

Combining these three issues in a sub-screen with a login portal would show a user how dangerous they can be and how secure and safe authentication would be critical for both the company and the user's data.

The login portal inside the subscreen is simple – two input fields for the username and the password and two buttons. The buttons allow the tester to attempt login and registration (**Figure 2.5.1**). The user can exploit this vulnerability by taking advantage of the detailed error messages and brute-forcing their way into the application's database manually or with the use of automated tools such as Hydra. Additionally, due to the lack of sanitisation and encryption, the user can use SQLi attacks to dump user login details.

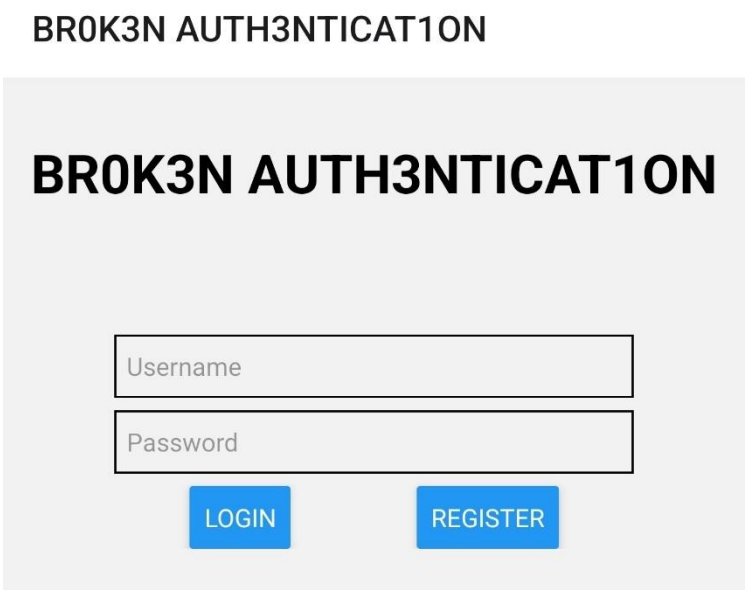

Figure 2.5.1 *– Broken Authentication screen.*

2.6 CROSS-SITE SCRIPTING (XSS)

The last vulnerability implemented by the team was cross-site scripting. This was possible due to the JavaScript nature of the framework. It already has several safety features against XSS, whilst a workaround was required. (Snyk, 2020)

The developers rendered a WebView within the application. The WebView provided the programmers with the ability to use HTML tags as if they were creating a website. This allowed them to input JavaScript within the input field without React Native's safety features interfering with the malicious code. Additionally, an SQLite database was used to save the user input, permitting Stored XSS attacks. The developers used this workaround method for one more reason – a lot of companies would rather reuse old, functional code than pay developers to write a new one. Such development will open the applications to many issues in terms of the software's security and could be fatal for both the users and the organisation.

Similar to the weakness from the previous section, this vulnerability was implemented into a simple subscreen with a text input field, a button used to send the malicious code to the database, and a field displaying the results of running the code. The vulnerability can be exploited by taking advantage of the WebView implementation and storing malicious code. The level of the threat will vary with the script – from harmless alerts to stealing credentials and sensitive company data.

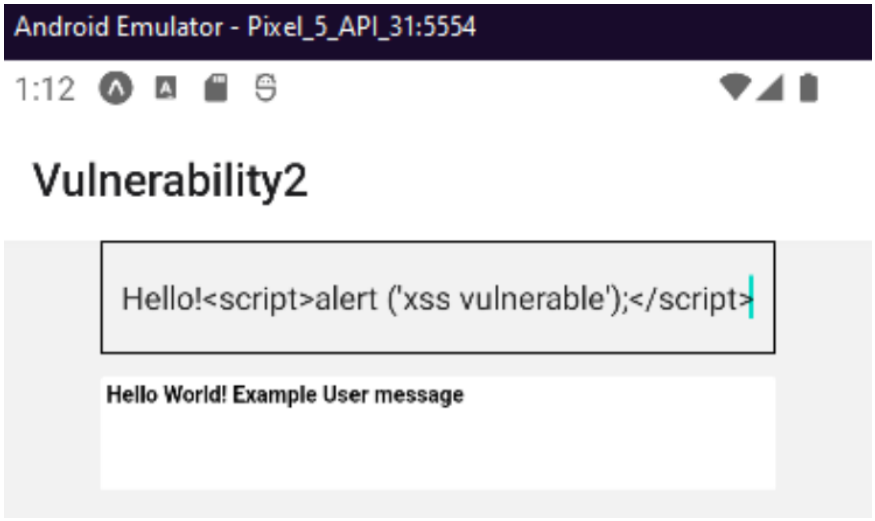

Figure 2.6.1 *– Cross-site Scripting Screen*

REFERENCES

Boughdiri, A. (2021). *React Native CLI vs Expo.* Available: https://blog.devgenius.io/react-native-cli-vs-expo-ab96dbc76cdd. Last accessed 28th Jan 2022.

ReactNavigation. (2016-Present Day). Bottom Tabs Navigation. Available: https://reactnavigation.org/docs/bottom-tab-navigator/. Last accessed 15th Feb 2022.

ReactNative. (2022). *Security.* Available: https://reactnative.dev/docs/security. Last accessed 24th Feb 2022.

React Native. (2022). Enabling Proguard to reduce the size of the APK. Available: https://reactnative.dev/docs/signed-apk-android#enabling-proguard-to-reduce-the-size-of-the-apk-optional. Last accessed 4th Mar 2022.

Snyk. (2020). Cross Site Scripting (XSS). Available: https://security.snyk.io/vuln/SNYK-JS-REACTNATIVEWEBVIEW-1011954. Last accessed 10th Mar 2022.

gnardini. (2020). Universal XSS in Android WebView. Available: https://github.com/react-native-webview/react-native-webview/issues/1655. Last accessed 11th Mar 2022.

secureITmania. (2021). *Let's know How I have explored the buried secrets in React Native application.* Available: https://infosecwriteups.com/lets-know-how-i-have-explored-the-buried-secrets-in-react-native-application-6236728198f7. Last accessed 24th Mar 2022.

bongtrop. (2021). React Native Application Static Analysis. Available: https://suam.wtf/posts/react-native-application-static-analysis-en/. Last accessed 24th Mar 2022.

APPENDICES

APPENDIX A – NAVIGATION HANDLING CODE

```
import { StatusBar } from 'expo-status-bar';
import * as React from 'react';
import { StyleSheet, Button, Text, View, Alert } from 'react-native';

import {NavigationContainer} from '@react-navigation/native';
import { createNavigationContainerRef } from '@react-navigation/native';
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';
import Ionicons from 'react-native-vector-icons/Ionicons';

// Screens

import HomeScreen from './screens/HomeScreen';
import VulnScreen from './screens/VulnerabilitiesScreen';
import DocScreen from './screens/DocumentationScreen';
import Vuln1Screen from './screens/Vuln1Screen';
import Vuln2Screen from './screens/Vuln2Screen';
import Vuln3Screen from './screens/Vuln3Screen';
import AboutUs from './screens/AboutUs';
import MoreInfo from './screens/MoreInfo';

// Screen names
const homeName = "Home";
const vulnName = "Vulnerabilities";
const docName = "Documentation";
const vuln1Name = "Vulnerability1";
const vuln2Name = "Vulnerability2";
const vuln3Name = "Vulnerability3";
const aboutUs = "About Us";
const moreInfo = "More Info";


const Tab = createBottomTabNavigator();
const navigationRef = createNavigationContainerRef();

export function navigate(name, params) {
    if (navigationRef.isReady()) {
      navigationRef.navigate(name, params);
    }
    else {
        navigationRef.current.getRootState()
    }
```

```
  }

export default function MainContainer(){
    return(
        <NavigationContainer ref = {navigationRef}>
            <Tab.Navigator
            initialRouteName={homeName}
            screenOptions={({route}) => ({
                 tabBarStyle: {height: 60, paddingTop: 7},
                tabBarIcon: ({focused, color, size}) => {
                    let iconName;
                    let rn = route.name;

                    if (rn === homeName) {
                        iconName = focused ? 'home' : 'home-outline';
                    }
                    else if (rn === vulnName) {
                        iconName = focused ? 'terminal' : 'terminal';
                    }
                    else if (rn === docName) {
                        iconName = focused ? 'ios-file-tray-full-sharp' : 'ios-
file-tray-full-sharp';
                    }

                    return <Ionicons name={iconName} size={size} color={color}/>
                },
            })}

            tabBarOptions={{
                //tabBarStyle: {padding: 10, height: 200},
                activeTintColor: 'orange',
                inactiveTintColor: 'grey',
                labelStyle: {paddingBottom: 10, fontSize: 10},
            }}
            >
            {/*Visible tab Screens*/}
            <Tab.Screen name={homeName} component={HomeScreen}/>
            <Tab.Screen name={vulnName} component={VulnScreen}/>
            <Tab.Screen name={docName} component={DocScreen}/>



            <Tab.Screen  name = {vuln1Name} component = {Vuln1Screen} options={{
            tabBarButton: () => null,
            tabBarVisible: false,
```

```
            title: 'BR0K3N AUTH3NTICAT1ON'
            }} />
            <Tab.Screen  name = {vuln2Name} component = {Vuln2Screen} options={{
            tabBarButton: () => null,
            tabBarVisible: false,
            title: 'Stored XSS'
            }} />
            <Tab.Screen  name = {vuln3Name} component = {Vuln3Screen} options={{
            tabBarButton: () => null,
            tabBarVisible: false,
            title: 'WOW an API?'
            }} />
            <Tab.Screen  name = {aboutUs} component = {AboutUs} options={{
            tabBarButton: () => null,
            tabBarVisible: false,
            }} />
            <Tab.Screen  name = {moreInfo} component = {MoreInfo} options={{
            tabBarButton: () => null,
            tabBarVisible: false,
            }} />

            </Tab.Navigator>

        </NavigationContainer>
    )
}

const styles = StyleSheet.create({
    container_main: {
      flex: 1,
      backgroundColor: '#fff',
      alignItems: 'center',
      paddingTop: Platform.OS === 'android' ? 65 : 0
    },

    titleText: {
      textAlign: 'center',
      fontSize: 30,
      fontWeight: 'bold',
    },

    normalText: {
      textAlign: 'center',
      fontSize: 20,
    },
```

```
    button_allign: {
      flex: 1,
      flexDirection: "row",
      justifyContent: "center",
      alignItems: "center",
    }
  });
```

APPENDIX B – VULNERABILITIES SCREEN CODE

```
import * as React from 'react';
import {View, ScrollView, StyleSheet, Text, Button, Alert} from 'react-native';

import * as Navigation from '@react-navigation/native';
import GlobalStyles from './stylesheets/styles';



export default function VulnScreen({navigation}) {
    return(

      // Main Scroll View
      <ScrollView style = {GlobalStyles.scrollContainer}>

        {/* Vulnerability 1 (Broken Authentication) */}
        <View style = {GlobalStyles.vulnPick}>
          <Text style = {GlobalStyles.heading}>
          Broken Authentication
          </Text>

          <Text style = {GlobalStyles.body}>
          This vulnerabilty is all about issues that can be found
          within the Authentication like for example the login page.
          </Text>
          <View style = {{ marginTop: 10, marginHorizontal: '25%'}}>
            <Button
              title = 'Broken Authentication'
              onPress = {() => navigation.navigate('Vulnerability1')}
            />
          </View>
        </View>
```

```
      {/* Vulnerability 2 (XSS) */}
      <View style = {GlobalStyles.vulnPick}>
        <Text style = {GlobalStyles.heading}>
        Cross-site Scripting (XSS)
        </Text>

        <Text style = {GlobalStyles.body}>
        This vulnerabilty is about injecting malicious JavaScript code into
        fields that do not have sufficient input validation.
        </Text>

        <View style = {{ marginTop: 10, marginHorizontal: '25%'}}>
          <Button
            title = 'Cross-site Scripting'
            onPress = {() => navigation.navigate('Vulnerability2')}
          />
        </View>
      </View>


      {/* Vulnerability 3 (SQL Injection) */}
      <View style = {GlobalStyles.vulnPick}>
        <Text style = {GlobalStyles.heading}>
        WOW an API?
        </Text>
        <Text style = {GlobalStyles.body}>
        WOW an API will introduce you to a data leakage vulnerability
        and the ease of reverse engineering React Native applications without
        appropriate countermeasures.
        </Text>
        <View style = {{ marginTop: 10, marginHorizontal: '25%'}}>
          <Button
            title = 'WOW an API'
            onPress = {() => navigation.navigate('Vulnerability3')}
          />
        </View>
      </View>
    </ScrollView>
    );
};
```

APPENDIX C – DATA LEAKAGE CODE

```
import React, { useState} from 'react';
import {View, Text, Button, ScrollView, TextInput, Alert, Linking } from 'react-
native';
import { Dialogflow_V2 } from "react-native-dialogflow"
import { Dialogflow } from 'react-native-dialogflow/js/Dialogflow';
import GlobalStyles from './stylesheets/styles';
class DialogFlow extends Dialogflow {
constructor(props) {
    super(props);

        Dialogflow_V2.setConfiguration(
            "your-dialogflow-project@asdf-76866.iam.gserviceaccount.com",
            '-----BEGIN PRIVATE KEY-----
\nMIIEvgIBADAN...1oqOawdddddiojdq9wq09wd81u1ijzcjkllasd\n-----END PRIVATE KEY----
-\n',
            Dialogflow_V2.LANG_ENGLISH,
            'testv2-3b4dss',
            'FLAG{WOW_API}'
        );
    }
}




export default function Vuln3Screen(navigation) {
    url =
'https://drive.google.com/file/d/1Q6IVqhCVi9GbxPzv6p85xC4SbBNb70b2/view?usp=shari
ng'

    const [flag, setFlag] = useState('');

    function checkFlag() {
        if (flag.length == 0) {
            alert ('Field is empty! Please input the flag.')
        }
        else if (flag == 'FLAG{WOW_API}'){
            Alert.alert('Victory!',
                        'Congratulations, you found the flag!')
        } else {
            Alert.alert('Oh no!',
                        'Invalid flag. Please try again.')
        }
    }
```

```
    return(
    <ScrollView style = {GlobalStyles.container}>
        <View style = {GlobalStyles.flex1}>
            <Text style = {GlobalStyles.title}>
                WOW an API?
            </Text>
            <View style ={GlobalStyles.loginStylesheet}>

                <TextInput
                    style={GlobalStyles.input}
                    placeholder="Please enter the Flag"
                    onChangeText={(flag) => setFlag(flag)}
                />
                <View style={{height: 30}} />

                <View style = {GlobalStyles.buttonView}>
                    <Button
                        title = 'Hint'
                        style = {{}}
                        onPress = {() => Alert.alert( 'Hints',
                                                '1. Perhaps the archives are
complete? \n2. Bundle your power. \n3. Seek the flag and ye shall find.')}
                    />
                    <View style={{width: 70}} />
                    <Button
                        title = 'Submit'
                        onPress = {checkFlag}
                    />
                </View>
                <View style={{height: 30}} />
                <View style = {GlobalStyles.buttonCenter}>
                <Button
                        title = 'Walkthrough'
                        style = {{}}
                        onPress = {() => Linking.openURL(url).catch((err) =>
{console.log(err)})}
                    />
                </View>
            </View>
        </View>
    </ScrollView>
    );
}
```

APPENDIX D – GITHUB BRANCHES AND ACTIVITY

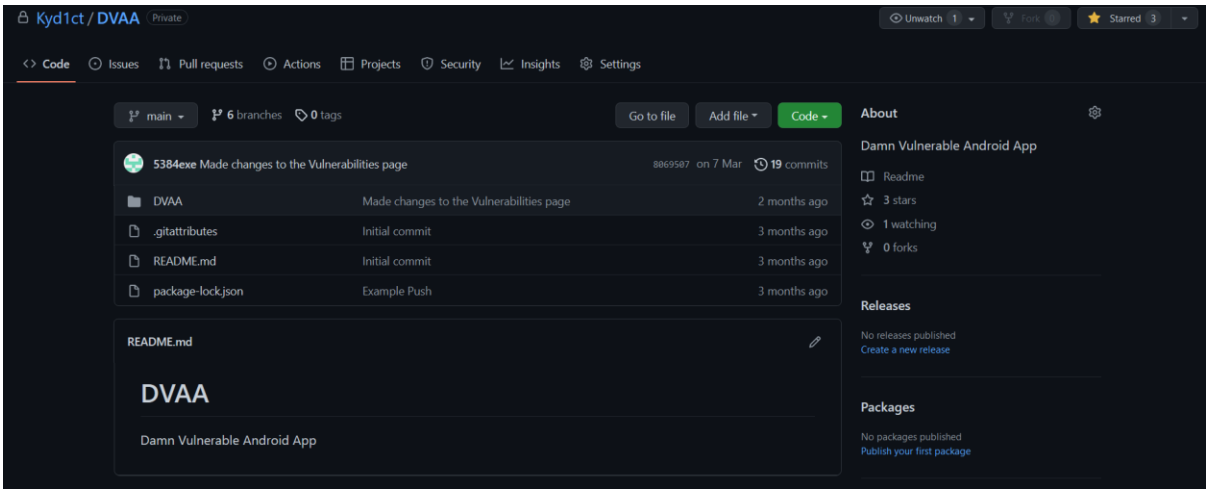

Figure 1 – Main branch and repository page.

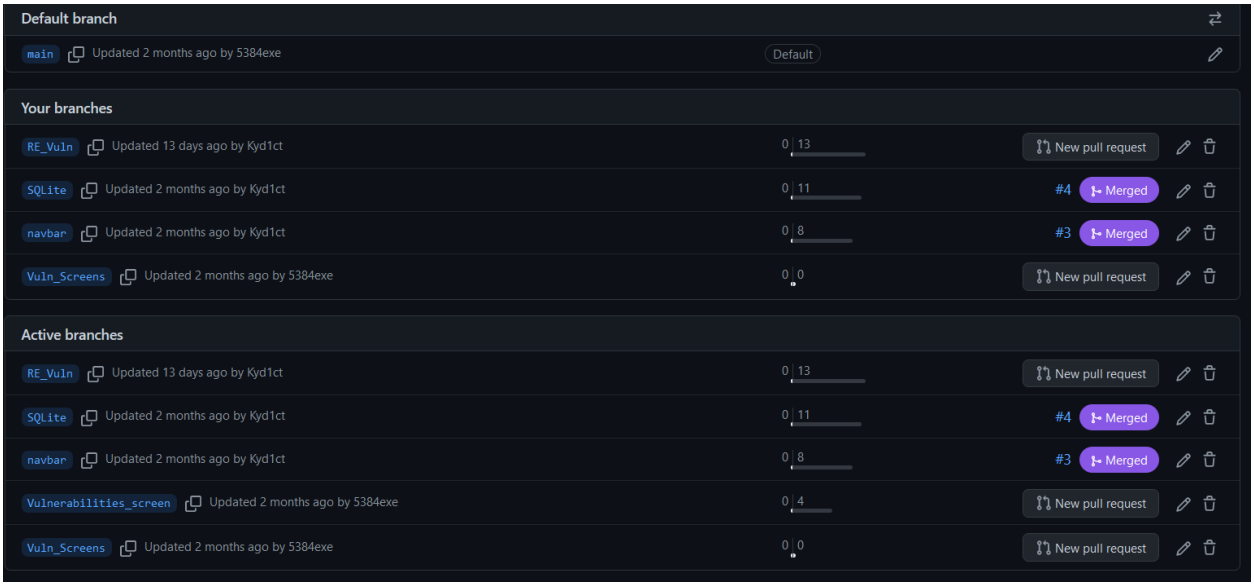

Figure 2 – Branch activity.

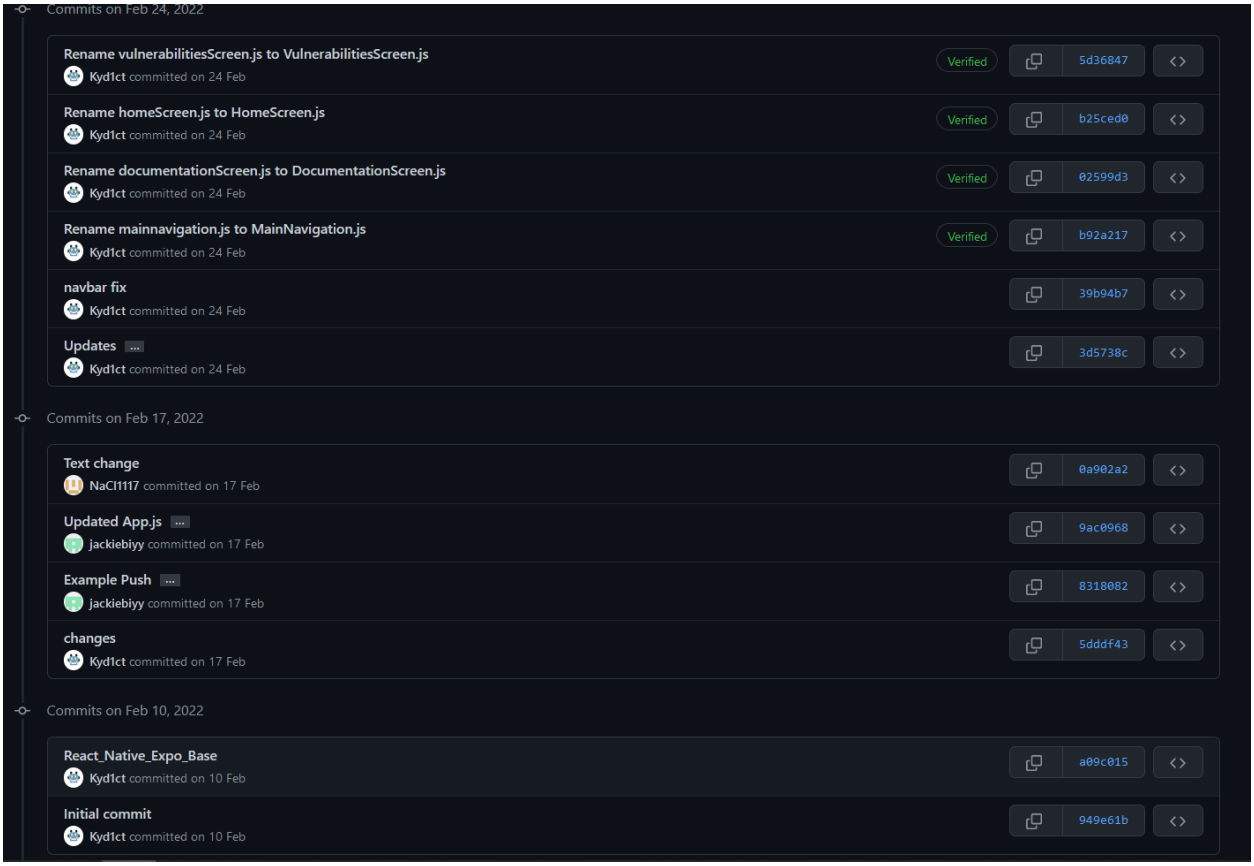

Figure 3 – Commits from 10th Feb 2022 to 24th Feb 2022 (Main branch pull request)

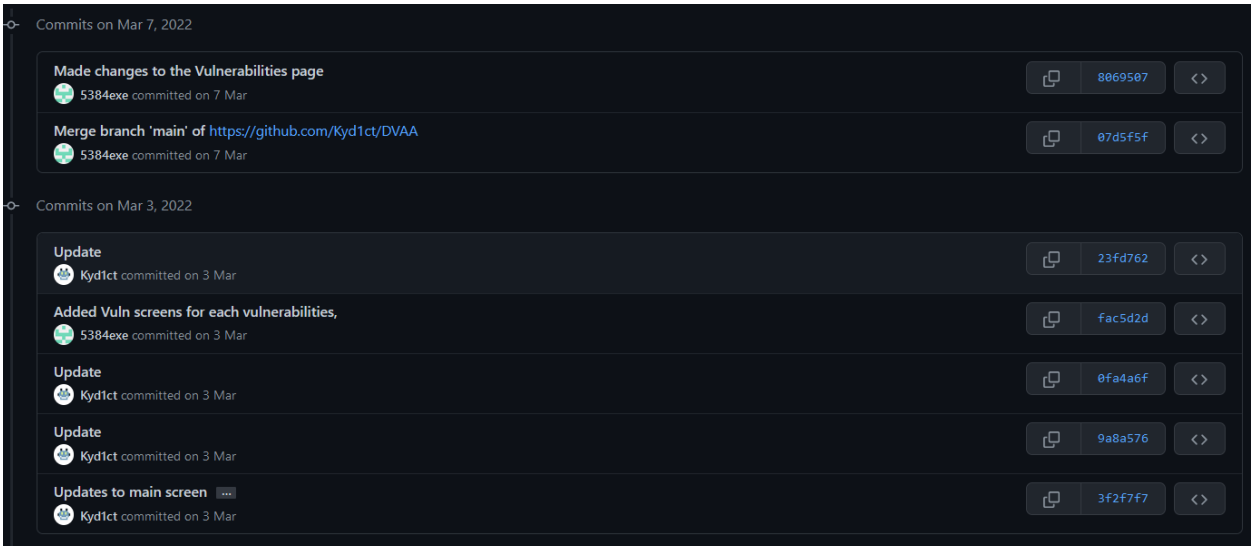

Figure 4 – Commits from 3rd Mar 2022 to 7th Mar 2022 (Main branch pull request)

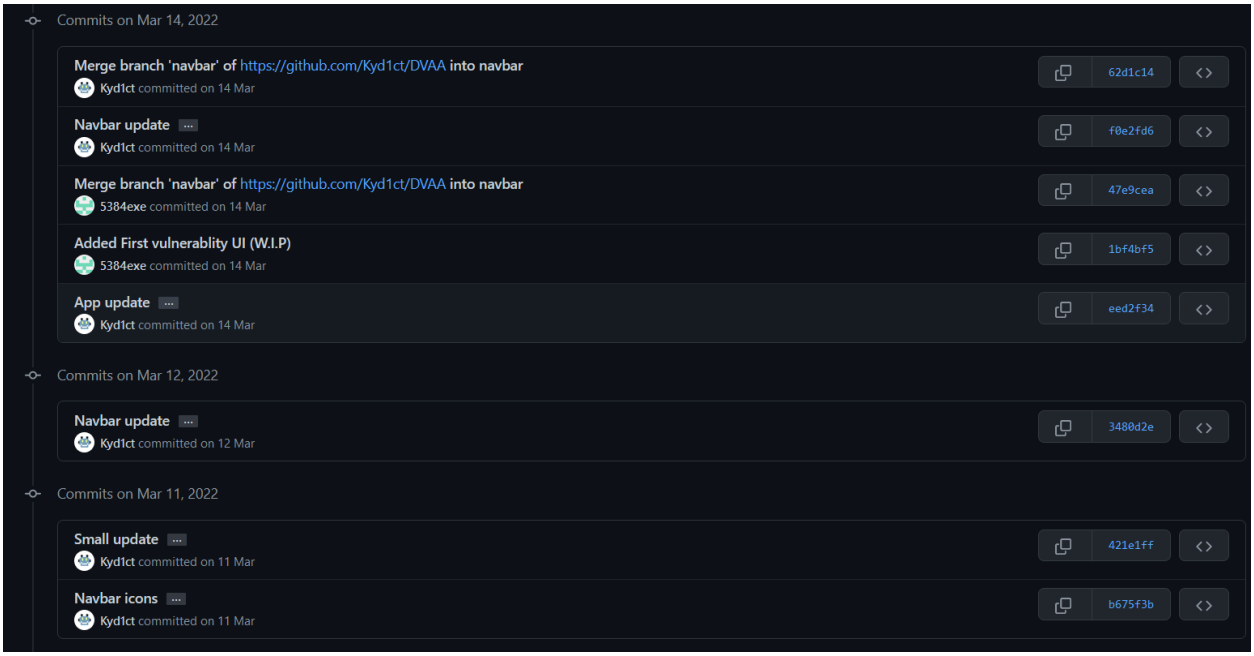

Figure 5 – Unmerged commits from 11th Mar 2022 to 14th Mar 2022

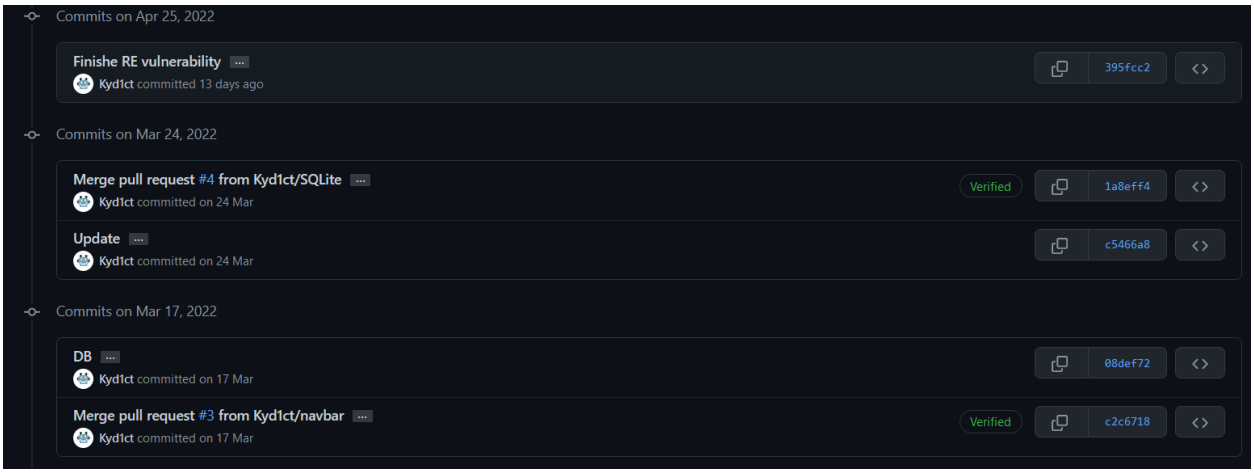

Figure 6 – Latest unmerged commits (17th Mar 2022 – 25th April 2022)

APPENDIX E - DELIVERABLES & REQUIREMENTS (REQUIRED)

Agreement Form: Project Deliverables

Group Number, Names of Team Members and Programme	Ethical Hacking 10 Martin Georgiev BSc Ethical Hacking
Programme specialist's Name:	Ethan Bayne
The deliverables listed below will be submitted by the team by the due date:	
Part A deliverables	To be agreed by programme specialist and team: • Executable code (Android APK) • Development documentation • Vulnerability documentation
Programme specialist's signature:	A.U. EBMT
Team member's signatures:	
Martin Georgiev	Martin Georgiev

Agreement Form: Requirements

Group Number: Ethical Hacking 10

Team Members: Sebastian Kielich, Scott Grant, Jack Gordon, Martin Georgiev

Project Title: Damn Vulnerable Android App

Please refer to the attached documentation for full details of the project. The requirements are listed in Table 1. The signatures below indicate that the requirements for this project have been agreed by the project stakeholder.

Any changes to the documentation should be made using the correct change authorisation procedure agreed with the programme specialist.

Table 1:

ID	List of Agreed Requirements
1	Responsive User Interface
2	Three Vulnerabilities
3	Documentation on the vulnerabilities – including explanations of why they occur
4	Vulnerability mitigation documentation – code snippets for vulnerable and secure code comparison, explanation of both snippets
5	Guidance for vulnerability exploitation (Hints and walkthrough)

Team members:	Signatures:	Date:
Martin Georgiev	Martin Georgiev	03/02/2022

Subject Specialist	A.U. EBMT	2022-03-03
Client (if applicable)	A.U. EBMT	2022-03-03

APPENDIX F – OTHER (OPTIONAL)

Image 1 – DVAA's application Icon

[1] Document prepared and revised by Natalie Coull, Colin McLean, Andrea Szymkowiak

[2] Graham, G. (2005). *The White Paper FAQ (Frequently Asked Questions)/That White Paper Guy – Gordon Graham*. [online] Available at: https://www.thatwhitepaperguy.com/white-paper-faq-frequently-asked-questions/#what_is (Accessed: 9 May 2016).

[3] Outline adopted from Gormandy-White, M. (2021) How to Write a User Manual (That's Easy to follow). Available at: https://grammar.yourdictionary.com/grammar-rules-and-tips/tips-on-writing-user-manuals.html (Accessed: 12 December 2021).