Android Applications with Secure Coding

PragneshPatel[1],Mihir Mishra[2], Kushal Patel[3],Pankti Naik[4],Patel Vibhuti[5]

Computer Engineering
GIDC Degree Engineering College, Navsari,Gujarat

Abstract—**Today Smart phones popularity is much more than ever. The main reasons for this is the fact that the Android operating system (OS) developed by Google is an open source platform that allows developers to write applications and distribute them for free in an open market. It is extremely essential for developers to understand how they can create secure Android applications. In this paper we are primarily focuses on secure coding practices for Android applications. This paper covers introduction to major components of Android application like Activity, Services Broadcast Receiver, Content Provider and Intents. It also includes secure coding recommendation and associated client-side application attacks which will help developers to create more secure android applications in future.**

Keywords— **Android, Operating System (OS), Activity, Broadcast Receiver, Content Provider, Intent.**

I. ANDROID APPLICATION COMPONENTS[1]

Before discussing Android's security options, it is necessary to take a basic overview of the major components of an Android application to understand how the different pieces fit together. Different components of android application are as follows.

A. *Activity*

In Android development terms, an "Activity"represent a single screen with user interface just like window or frame in java that interacts with a user and provides functionality. An activity forms the essential building blocks of the application.

B. *Intent*

Intent is an abstract description of an operation to be performed. It is commonly used to start an activity or service. Intents can be used to broadcast and received within the application itself and with other applications.

C. *Service*

A service is a component that runs in background to perform long running operation without needing to interact with users. These operations are performed without affecting the main application running on the front end. Service continues to run in background, even when application is destroyed.

D. *Content Providers*

Content providers store data persistently. Content provider supplies data from one application to others on request. Content providers also provide the best means to share data between applications. Content provider can use different ways to store data and data can be stored in database and files or on the network.

E. *WebView*

WebViews act like a web browser to display HTML content to the user.WebView makes turns your application into a web application. WebView uses WebKit engine to display web pages.If there is any vulnerability inWebKit it directly impacts the WebView. This component allows a user to navigate forward and backward through the history, zoom in and out, and perform text searches, just like Internet Explorer, Firefox, or any other browser.

F. *Permissions*

The primary security of an application is defined by its permissions. The level to which an application can perform an action is limited to the permissions defined in its AndroidManifest.xml. By default, every application is sandboxed by the OS and restricts access to the data of another application. At the time of application installation, the user is presented with the list of permissions that are required by the application. Once the user grants those permissions, only then the application will be installed.

II. SECURE CODING SUGGESTIONS[4]

In the previous section, we have covered the various Android components used in an application. In this section, we will cover secure coding guidelines and the associated client-side application threats.

A. Lock-down application permissions

It is essential to follow the principle of least privilege when assigning permissions. Permissions should not be assigned until they are required. The application should grant only the minimum required permissions at the architecture level. For example, both READ and WRITE permissions should not be granted when only READ permissions are required. This is the most common mistake made by developers due to a lack of understanding of the functionality of an application and proper knowledge.

For example, the Android:protectionLevel element in the AndroidManifest.xml file shows the protection/risk level associated with the installed application. When the value of the parameter is dangerous, then when application is installed, it gains permission to access user data and to control the device. Developers should take extreme care while assigning applications with high-risk permissions.

B. File permissions

File permissions apply to files stored on external storage. If developer has created file using openFileOutput then that file is private to the application and cannot be accessed by other applications. Developer should pay close attention before providing a file with the MODE_WORLD_READABLEor MODE_WORLD_WRITABLE permissions, because it allows other applications to access the file. So developer should not provide the writable option until it is required to enforce the principle of least privilege.

There are two levels for defining permissions, in the AndroidManifest.xml file and in program's code. Both cases need to be analysed for security issues.

1. Permission defined in AndroidManifest.xml are as below:

```
<uses-permission android:name=”android.permission.INTERNET” />

<uses-permission android:name=”android.permission.SEND_SMS” />
```

2. In the code permission defined for file is as below:

```
String OUTPUT_FILE = “file.txt”;

FileOutputStreamfos = openFileOutput(OUTPUT_FILE,Context.MODE_WORLD_WRITEABLE);
```

C. Custom permissions

Custom permissions are described to the user via a permission description within the Android:description tag of AndroidManifest.xml. Proper Care should be taken to ensure the description is ample to provide the layperson user with information that details what permission is being granted. Sample code showing the description defined for custom permission used:

```
<permission
android:name=”com.testpaccourierkage.mypermission
android:label=”my_permission” android:protectionLevel=”dangerous”
android:description=”@string/detonate_description”>
</permission>
```

D. Protect pending intents

The pending intent is basically an object that wraps another Intent object.PendingIntent can be passed to foreign application where you are granting the app the right to perform operations. The pending intents allow the intent in your application to be invoked by another application. Just invoking the intent is not the problem. The problem is that the application that invokes the intent also executes at the same permission level as that of the application that had the pending intent to be invoked. By allowing other applications to invoke PendingIntent in our application, we are allowing the other applications to execute at the same privilege as that of our application. Hence, we should always pass explicit intents to a PendingIntent rather than being implicit.

Below code snippets shows checking for secure usage of pending intent—you can identify that intent is being handed over to the alarmManager application:

```
Intent myIntent = new Intent(AndroidAlarmService.this,MyAlarmService.class);
pendingIntent=PendingIntent.getService(AndroidAlarmService.this, 0,myIntent, 0);
AlarmManageralarmManager =
(AlarmManager)getSystemService(ALARM_SERVICE);
Calendar calendar = Calendar.getInstance();
calendar.setTimeInMillis(System.currentTimeMillis());
calendar.add(Calendar.SECOND, 10);
alarmManager.set(AlarmManager.RTC_WAKEUP,calendar.getTimeInMillis(),pendingIntent);
```

Once identified, check whether intent is handed over to a third-party application. If it is true then try to figure out what the third-party application does with the pending intent (that is, alaramManager). Here it is necessary to understand that the trust is based on the third-party application to determine if any security issues exist. The best way to deal with PendingIntentis If a PendingIntent is an application requirement, make sure that only a trusted application receives it. This leaves no room for that intent to be used by a mistrustful application.

E. *Content providers*

Content Provider provides the mechanism that is used to share persistent data across applications. By default, ContentProviders can be invoked and accessed by any foreign application if permission is not set with it. If permission is set on ContentProvider in AndroidManifest.xml file, then only those applications that are granted permission can access that content provider.

We can define both read and write permission as needed. This is shown in the declaration below:

```
<providerandroid:name=”MyContactsProvider”
android:authorities=”com.permission.test”
android:readPermission=”android.permission.permRead”
android:writePermission=”android.permission.permWrite”/>
```

F. *Insecure Storage*

Android Application data can be stored in two folders:

- /data/data/<package name of application>
- /sdcard

From the security point of view data stored in both of these directories behave differently. Data thatis stored in /data/data/<package name of application> cannot be accessed by anotherapplication unless the application openly provides permissions or if the Android device is rooted.Data that is stored in /sdcard can be accessed by any application without the need for any specialpermission or rooting. Hence, it is common for foreign applications to access data in /sdcard.Developer shouldtry to minimize situations whensensitive data that needs to be stored on the device. If any data needs to be stored, then encrypt itusing a strong encryption algorithm before to storage.

G. *PreferenceActivity storage*

SharedPreference allows a user to store data locally that can be read by all the classes in theapplication. By using SharedPreferencesan XML file is created in /data/data/<Package name ofapplication> and data stored in key-value pair. This data persists even after the application is closed. If the mobile device is stolen,then data stored within the XML file can be easily retrieved by rooting the device.

H. *Improper usage of WebView[4]*

The WebView class is one of the most powerful classes, and it is used to display web pages inside a normalbrowser. It also allows applications to interact with WebView by adding a hook, monitoring changesbeing made, add JavaScript, and more. Though this a great feature, it creates security loopholes if not used properly. Since WebView can be customized, it creates thechance to break out of the sandbox and bypass the same origin policy.

Through WebViewsandbox can be bypass in two different scenarios:

- JavaScript can invoke Java code.
- Java code can invoke JavaScript.

1. Sample code to Invoke Java from JavaScript:

```
wv.addJavascriptInterface(new FileUtils(), "file");
<script>
filename = '/data/data/com.Foudnstone/data.txt';
file.write(filename, data, false);
</script>
```

2. Sample code to Invoke Java script from java

```
String javascr = "javascript: varnewscript=document.
createElement(\"script\");";
javascr += "newscript.src=\"http://www.foundstone.com\";";
javascr += "document.body.appendChild(newscript);";
myWebView.loadUrl(javascr);
```

Now consider a situation where there was link in the WebView that redirected a user to a maliciouswebsite or to load malevolent JavaScript from another website. Attacker-controlled code can usegenuine sandbox bypass code similar to the examples above and access all application data.

For providing security while using theWebView, developer should look for the action that is being performed while the loadurlevent is triggered. There are two scenarios where the security issue can be found.

- Loadurl event is not overridden.
- Loadurl event is overridden and the user is not restricted to the base URL of theapplication in Loadurl event.

To avoid security issues from the WebView, always limit users to the application area using thecode shown below.

```
WebViewclientwvclient = New WebViewClient() {

// override the "shouldOverrideUrlLoading" hook.

publicbooleanshouldOverrideUrlLoading(WebViewview,Stringurl){
if(!url.startsWith("http://clientlocation.com")){
Intent i = new Intent("Android,intent.action.VIEW",Uri.parse(url));
startActivity(i);
// override the "onPageFinished" hook.
public void onPageFinished(WebView view, String url) { ...}
}
webView.setWebViewClient(wvclient);
// override the "onPageFinished" hook.
public void onPageFinished(WebView view, String url) { ...}
}
webView.setWebViewClient(wvclient);
```

Other suggestions that developer needs to take care while coding with WebView are listed as below:

- Do not call setJavaScriptEnabled() for WebView until there is need for processingJavaScript. If JavaScript is not enabled, then attacks like XSS, defacing, and others willbe eliminated.
- Compile the application against Android API level equal to or more than 17. This APIforces the developer to add @JavascriptInterface to any method that will be exposed toJavaScript. This also prevents access to OS commands (via java.lang.Runtime).
- Send all traffic over SSL. Any traffic is easy to sniff and manipulate using a man-in-themiddleattack. A hacker cannot inject script via MITM and cannot break the sandbox ofWebView.

I. *Perform data validation*

Data validation issues in Android are generally not considered as serious during penetration testingor while performing a code review. But it is not good from developer point of view.As we know that WebView itself is a browser instance and has all the capabilities ofa browser that make WebView more vulnerable to all browser attacks.An Android application can be writtenin Java or native code like C++. If Java is used, manyof the data validation issues like buffer overflow, format string issues, and others are eliminated, asthe language itself is not vulnerable. When using native code, special care needs to be taken whendata is read from an untrusted source because it is vulnerable to conditions like buffer overflow, formatstring issues, and more.When performing data validation code review, it is important to identify the source and the sink.Source refers to the place where the data is stored. Sink refers to the place where data is sent backto user. Once complete flow from the source to sink is understood, we can easily identify what kindsof issues there are, for example, XSS, SQL injection, buffer overflow, and many more data validation relatedissues.

J. Common mistakes

Other common issues that occur when Android applications are being developed and that are similarto the web application penetration testing are mentioned below:

- Mostly logout functionality of a user in an Android application is generally done only on the client side bymoving him from to new screen. The application should also send an explicit logout request tothe server to terminate server-side sessions.
- Session ID or sensitive data should not be sent in request URLs.Poor error-handling leads to sensitive information disclosure.
- Data entered through mobile applications, including Android applications, is often stored inthe backend servers without validation. The stored data is then accessed via the webapplication, resulting in security issues like XSS, malicious URL injection, and others.
- Android applications are also vulnerable to web-related attacks like XSS, CSRF, XFS, andothers when WebView is used.
- Session ID timeout is usually very long or sessions do not expire. Invalidate session Idson both the client and server side after 30 minutes of inactivity.

III. Conclusion

In this paper we have studied fundamental android application components and secure coding recommendations associated with those components.As there is astonishing growth in number of Android devices activated per day it is responsibility of developer to create more secure android applications. By referring this paper developers will be aware of attacks that can be occurs on their applications and developer can try to improve their coding skills in context of security.

References

[1] http://developer.Android.com/

[2] William Enck, Damien Octeau, Patrick McDaniel, and SwaratChaudhuri,” A Study of Android Application Security”,Department of Computer Science and Engineering, The Pennsylvania State University.

[3] Johnson, Ryan, et al. "Analysis of android applications' permissions."Software Security and Reliability Companion (SERE-C), 2012 IEEE Sixth International Conference on. IEEE, 2012.

[4] Naveen Rudrappa,”Secure coding for Android Applications ”, senior security consultant at McAfee Foundstone Professional Services.

[5] Meier, Reto. Professional Android 4 application development. John Wiley & Sons, 2012.

[6] Felt, Adrienne Porter, et al. "Android permissions demystified." Proceedings of the 18th ACM conference on Computer and communications security. ACM, 2011.