

# Security Analysis Protocol for Android-Based Mobile Applications

José Palacios<sup>1</sup>, Gabriel López<sup>2</sup>, Franklin Sánchez<sup>3</sup>

jose.palacios01@epn.edu.ec, gabriel.lopez@epn.edu.ec, franklin.sanchez@epn.edu.ec

<sup>1,2,3</sup> Escuela Politécnica Nacional, 170525, Quito, Ecuador.

Pages:366–378

**Abstract:** This document presents a testing protocol to evaluate security of mobile applications based on Android. For this, it's been used as a reference the methodology proposed by OWASP Mobile Security Project. It's been developed a protocol of tests that consists of three phases: information gathering, static analysis and dynamic analysis. To assess the security of mobile applications the testing protocol has been designed, using certain steps of the OWASP Mobile Security Project. For the three phases that are part of the designed test protocol, a set of tasks is defined with the necessary procedure and tools. Subsequent to the design of the test protocol, it has been validated on a mobile application, and it has been presented comments of the test. The tools used in the protocol include dex2jar, PeaZip, axml2printer, jd-gui, logcat, OWASP ZAP and Drozer.

**Keywords:** Android; data gathering; dynamic analysis; mobile application; security; static analysis; OWASP.

## 1. Introduction

With the increasing use of mobile applications, a lot of enterprises have chosen to offer the possibility of making commercial transactions from the convenience of a mobile device. However, given the sensitivity of the data that is handled by these types of applications, it has become necessary to analyze the level of security of these applications in order to protect the customer's information.

The OWASP Mobile Security Project methodology (Owasp.org, 2016), has been used to design a testing protocol, which consists of three phases: data collection, static analysis and dynamic analysis. While the OWASP Mobile Security Project defines the points to consider for testing the security of mobile applications, it does not define the procedure or tools required for testing. Because of this, in this work a test protocol is defined for Android applications, which defines the procedure and tools needed for each of the tasks that are part of the phases of data collection, static analysis and dynamic analysis.

The analysis of Android applications for devices has already been studied in other investigations, such as: Mobile Application Security on Android: Context on Android security (Burns, 2009), documents like this one, in which only the theoretical approach is discussed, which serves as the basis for an analysis of Android applications, but does

not indicate steps or current tools to do it. Towards Black Box Testing of Android Apps (Zhauniarovich, Philippov, Gadyatskaya, Crispo & Massacci, n. d.). In which they focus on the design of a framework for Black Box analysis, for Android applications. However, they do not take into account the analysis of the application code components or the tools needed to perform the analysis.

A test protocol has been designed in order to test the security of mobile applications, using certain steps of the OWASP Mobile Security Project. For the three phases that are part of the designed test protocol, a set of tasks is defined with the necessary procedure and tools. Subsequent to the design of the test protocol, it has been executed on a mobile application of Ecuador and it has been found that there are several weak points in this, including an alarming vulnerability. Finally, the results obtained from the execution of the test protocol for the mobile application was analyzed.

Using the protocol proposed in this paper, it is possible to have a notion of the security status of mobile applications. Through the implementation of this protocol, vulnerable points can be detected, in order to be able to correct them later and to protect the data of the clients.

The second section of this document presents the related work, the third section presents the proposed protocol, along with the tools and commands required for its use. The fourth section shows the validation of the test protocol for a mobile application of Ecuador. Finally, the sixth section presents the conclusions.

## **2. Related work**

With the aim of analyzing the security of mobile applications, work has been carried out such as: OWASP Mobile Security Project methodology (Owasp.org, 2016), Mobile Application Security on Android: Context on Android security (Burns, 2009), Towards Black Box Testing of Android Apps (Zhauniarovich et al., n. d.), Securing The Mobile Banking Channel (Entersekt, 2014).

OWASP Mobile Security Project methodology, is a resource whose purpose is to provide developers and security teams with the resources they need to build and maintain secure mobile applications. This project seeks to classify mobile security risks and provide development controls to reduce their impact or probability of exploitation. For this, OWASP defines a set of steps grouped into three phases: data gathering, static analysis and dynamic analysis. However, for all the steps described in each of the three phases, there is no detailed procedure of how to carry them out, which is why the proposed protocol details the points to be considered within each of the three phases: data collection, static analysis and dynamic analysis, the procedure and the tools required for the implementation of the protocol on mobile applications.

Mobile Application Security on Android: Context on Android security (Burns, 2009) details a complete theoretical approach, which can then be used to define a test protocol. For example, it takes into consideration Android's security model, permissions, attempts, activities, broadcasts, services, content providers, etc. However, no protocol is defined to test the security of Android-based mobile applications, nor the necessary tools. This is why this paper includes a selection of specific procedures and tools for testing.

Towards Black Box Testing of Android Apps (Zhauniarovich et al., n. d.) deals with the development of a framework for black box testing, i.e. without having access to the application's source code. For this, this framework uses tools such as: Apktool, dex2jar and Emma. However, it defines more of a set of tools than a test protocol with defined steps for assessing the security of an application. That's why this document offers a series of defined steps, which provides a guide to evaluate an Android mobile application.

Securing The Mobile Banking Channel (Entersekt, 2014), deals with ways to secure the communication channel with mobile banking applications, in order to encourage the use of these by customers. For this purpose, recommendations are provided that the developer of mobile banking applications should take into account during the design process. However, it does not define any procedure for defining the security of an application nor a set of defined steps can be followed. This is why this document offers a protocol, which provides a guide to determine the security status of an application, taking into account not only the communications channel but also its code.

In relation to the works mentioned above, it is found that they do not define a set of defined steps, which allow the security level of Android-based mobile applications to be determined. For this purpose, this document defines three phases: data gathering, static analysis and dynamic analysis, which have a set of defined steps, as well as the necessary tools to evaluate the security level of mobile applications based on Android.

### 3. Methodology

The figures and examples shown below were obtained from the implementation of the protocol on a mobile application.

The OWASP Mobile Security Project methodology was used as a reference for the analysis of applications. OWASP methodology mentions the points needed for the analysis, but it does not indicate the tools or steps to be taken in detail. This chapter describes the test protocol and the required tools. As mentioned above, the protocol consists of three phases: data collection, static analysis and dynamic analysis.

#### 3.1. Information Gathering Protocol

The following describes the protocol for data collection, which is then used for the analysis of applications. For this, applications must be obtained from the official store, this with the aim of avoiding the use of malicious or modified applications. The tasks which are part of this phase are: Functionality and workflow, Network interfaces and hardware components, Commercial transactions and interaction with other applications, and Application signing.

1. **Functionality and workflow:** At this point the application is installed on a device and navigate through it to see the options it provides.
2. **Network interfaces and hardware components:** When the application is installed and it is opened for the first time, it requests some permissions. It is necessary at this point to make a screenshot, in order to remember the permissions requested. Subsequently, the set of permissions should be listed, separating the permissions that provide access to the network interfaces with which interacts with the application.

3. Commercial transactions and interaction with other applications: Using the exploration in section 3.1.1, it should be taken into account the information about the customer displayed by the application, and list the business transactions it allows to conduct. Also, it should be checked what information is needed or what process is to execute these transactions. For example, if the transactions can be executed only by accessing the application or if it is needed to confirm them with some additional code.
4. Application signing: Each application is signed with a certificate obtained by the application developer, to verify the origin of the application, it is necessary to verify the data contained in this certificate. Two tools are used to obtain this information: Jarsigner and OpenSSL.

### 3.2.Static Analysis

For this phase the application is installed on a virtual Android device. For each task that makes up the static analysis, the use of each of the tools necessary for the analysis is described in the following sections. In addition, there are recommendations that will help to perform each task. The task for static analysis are: Obtaining the source code, Permissions in the archive AndroidManifest.xml, Framework, Libraries, Views, Data entry, Code used for authentication, Online/Offline authentication, Additional information to the User/Password, Authentication methods, Blocking, Unique authentication, Two step authentication, and Automated static analysis.

Name	Size	Packed Size	Modified
assets	16 023 238	4 489 462	
lib	9 548 700	4 008 227	
META-INF	100 756	33 766	
res	736 532	712 843	
AndroidManifest.xml	5 856	1 686	2017-09...
classes.dex	497 692	165 645	2017-09...
resources.arsc	34 236	34 236	1980-12...

Figure 1 – Viewing.apk files using PeaZip

1. Obtaining the source code: Once the application is installed on the Android virtual device, the Apk Extractor application is installed on it. Apk Extractor lists and extracts the application to be analyzed and obtains its .apk file. Clicking on the desired application extracts the .apk file and displays the path in which it was saved.

Now that the .apk file has been obtained, the PeaZip tool is used to view the elements contained in this file. As shown in Figure 1, the .apk file is a zip file of everything needed to make the application work. These files are then extracted into a folder to make them available for analysis.

Then, the code that composes the application is obtained using the Dex2Jar tool. To do this, execute the command `d2j-dex2jar.bat` from the directory of the `dex2jar-2.0` folder, executing the following instruction, where the second part of the command is the path to the `classes.dex` file obtained in the previous step.

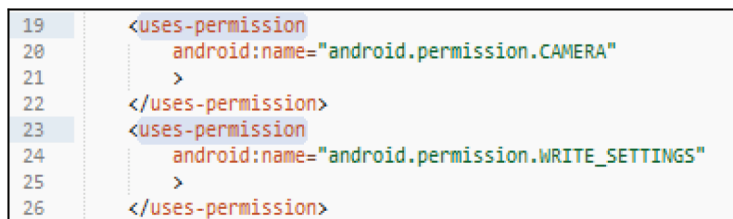
```
>d2j-dex2jar.bat C:\Users\josep\Desktop\classes.dex
```

The resulting file is located in the tool folder, this file will be used later, so it is recommended to put it in the same folder where the `.apk` file was extracted.

2. Permissions in the archive `AndroidManifest.xml`: To make this point you work with the `AndroidManifest` file.xml that was obtained in step 3.2.1. However, this file needs to be processed by another tool before it can be read, `AXMLPrinter2` is used for this. With the following command you get the readable file, the second part of the command indicates the name of the file where the result will be saved.

```
>java -jar AXMLPrinter2.jar AndroidManifest.xml > AndroidManifest_decoded_.xml
```

With the `AndroidManifest` file.xml already in a readable format, any text editor is used to read its contents. It is recommended to use the Sublime Text (Sublime Text, n. d.), tool as it allows a more comfortable reading of the file. The result is shown in Figure 2.



```

19      <uses-permission
20          android:name="android.permission.CAMERA"
21      >
22  </uses-permission>
23  <uses-permission
24      android:name="android.permission.WRITE_SETTINGS"
25  >
26  </uses-permission>
  
```

Figure 2 – Extract from the `AndroidManifest.xml` file

This file contains the permissions that the application requests at the time of its installation, these are found with the label `<uses-permission>`. These permissions should be listed to verify that they are the same as those the application requested when installing it. A detail of the most relevant permissions is shown in reference (Permissions Overview-Android Developers, n. d.).

3. Framework: This point aims to determine the framework in which the application was developed. The determination of this information is made according to the recommendations of the documents: (Quora, n. d.) or (Deitel University of London & Trinity College Dublin, 2012), where it is specified to navigate through the content of the decompressed `.apk` file, obtaining information about the framework used for the development of the application. In these documents it is recommended to search within the `assets` folder, if there is a `www` folder containing the name of packages such as those suggested in Owaps.org (n. d.), such as Apache Cordova (Apache Cordova, n. d.), Ionic (Ionic Framework, n. d.), etc. This information can also be found in the `src` folder.

4. External Libraries: Each application uses libraries that provide certain functionalities, these libraries can introduce vulnerabilities into the application, so it is necessary to identify them. For this purpose, the following procedure is performed.

Unzipping the .apk file finds the lib folder, this folder contains the libraries which are part of the application, for example, libraries that are not included in the Android SDK (Owaps.org, n. d.). For this, there is a folder for each processor architecture. Once identified the libraries which are part of the application, it can be used references such as: (NVD - Search and Statistics, n. d.), (CVE, n. d.), (Exploits Database, n. d.) o (US-CERT|United States Computer Emergency Readiness Team, n. d.) in order to look for known vulnerabilities in its database.

5. Views: From this point, it is necessary to mention that the code obtained through the previous procedure is not exactly the same as the original one that makes up the application, since a part of the code is lost during the process to create the .apk file (Godfrey, 2012).

Now that it is acquired the application code and the data obtained during the information gathering phase, it can be defined whether the application processes the user data, as the case with native and hybrid applications; or whether it only has the necessary views to connect to the server, and that it performs the entire process, as the case with web applications (Owaps.org, n. d.) . In the case of a native or hybrid application, when extracting the .apk file you will find the res folder, this folder contains the views of the application.

6. Data entry: JD-Gui is used for the analysis of the code, while Logcat is used for the analysis of the logs generated by the Android device at the time of using the application. This facilitates code analysis as logs can be displayed as the application is running, so that the code portions of each stage can be identified.

For data entry, the application is run in parallel with Logcat. It is recommended that each time it is performed a step, such as entering a user name or password, enter the log number so that it can be displayed it later, and thus know which logs belong to which stage.

For example, for this point the process of an electronic transfer is considered and the logs generated are. As shown in Figure 3, the logs generated show the Direct Transfer Confirmation Activity, and also the word requiresValidation which is later explored with the JD-Gui tool. To be able to successfully develop this point, it is necessary to collect the logs generated only by this particular process (for this example, an electronic transfer), and with that information search for the related code using the JD-Gui tool.

Now that the information from one of the Activities has been obtained, the words obtained in the previous point are searched in the application's .jar file, opened using the JD-Gui tool. As noted in Code 1, the phrase requiereValidation required has been found, obtained from the logs generated during the electronic transfer. Thanks to this it is found that the application performs validation of the data of the transferee, such as his name.

```
1 04-26 21:15:29.925 I/ActivityManager( 1838): START u0
    {cmp=com.application.application/.DirectTransferConfirmationActivity (has extras)}
    from uid 10063 on display 0
2 04-26 21:15:29.947 D/Volley ( 3406): [111] BasicNetwork.logSlowRequests: HTTP
    response for request=<[ ] https://sbapp07.application.com/
    application-mobile-services-pfm/xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
    0xe7aa217e NORMAL 10> [lifetime=8084], [size=946], [rc=200], [retryCount=0]
3 04-26 21:15:29.947 D/Response( 3406):
    {"resultadoValidaDestino":"requiereValidacion":true,"mensaje":
```

Figure 3 – Class identified in the logs generated during an electronic transfer

7. Code used for authentication: This section uses the information generated by the application and collected by Logcat to analyze the corresponding code. The authentication phase is the most important as the application must properly handle customer information so that sensitive information is not exposed.

At this point it must be searched for the code corresponding to the logs generated in the .jar file. Logs are displayed using Logcat while the application is running on the Android virtual appliance. With this it can be identified the methods that are executed and also the keywords that will be searched later by JD-Gui.

```

1 public ValidateDirectTransferParser(JSONObject paramJSONObject)
2 {
3     super(paramJSONObject);
4     try
5     {
6         Object localObject = paramJSONObject.getJSONObject("resultadoValidaDestino");
7         this.a = new DirectTransferSummary(parseProduct(((JSONObject)
8             localObject).getJSONObject("producto")));
9         JSONObject localJSONObject =
10             paramJSONObject.getJSONObject("resultadoDatosBeneficiario");
11         super(JSONObject) = localJSONObject.getString("nombreBeneficiario");
12         boolean bool = ((JSONObject)localObject).optBoolean("requiereValidacion");
13         this.a.setRequireValidation(bool);
14         if (bool)

```

Code 1 – ValidateDirectTransferParser class and the word found in the logs for an electronic transfer

8. Online/Offline authentication: To analyze this point, it is used the logs generated by the application, while it is running on the virtual appliance. It is required to know if the application connects with any external server to carry out the authentication (online), or if this process is carried out locally (offline).

In the case of an application that performs the online authentication process, URLs appear in the logs generated by the application to which it connects. As shown in Figure 4, this application makes use of the HTTPS protocol to connect to an external server during execution.

If the application is authenticated offline, the application does not connect to any URLs to provide access. Figure 5 shows an extract of the logs generated by an application that is authenticated offline, showing that there is no connection to an external server.



```

1 04-23 20:36:43.412 W/json ( 2452): https://www.aplicacion.com.ec/detectrd/
  image.htm?id=XXXXXXX
2 04-23 20:33:56.964 W/json ( 2452): http://www.aplicacion.com/Files/MovilApp/
  BancoApp.jpg

```

Figure 4 – Extract of the logs generated by an application that performs online authentication

9. Additional information to the User/Password: Within the authentication process, applications use a username and password, at this point it will be asked to find out if the application uses additional information. For example, there are applications that use a device identifier, coordinates, etc.

```

1 04-26 22:09:55.926 I/ActivityManager( 1838): START u0 {act=android.intent.action.MAIN
  cat=[android.intent.category.LAUNCHER] flg=0x10200000
  cmp=com.android.music/.MusicBrowserActivity (has extras)} from uid 10007 on display 0
2 04-26 22:09:55.927 W/AudioTrack( 1838): AUDIO_OUTPUT_FLAG_FAST denied by client
3 04-26 22:09:55.972 E/libprocessgroup( 3956): failed to make and chown /acct/uid_10033:
  Read-only file system
4 04-26 22:09:55.978 I/ActivityManager( 1838): Start proc 3956:com.android.music/u0a33
  for activity com.android.music/.MusicBrowserActivity
5 04-26 22:09:56.008 I/ActivityManager( 1838): START u0 {act=android.intent.action.PICK
  dat= typ=vnd.android.cursor.dir/artistalbum flg=0x4000000
  cmp=com.android.music/.ArtistAlbumBrowserActivity (has extras)} from uid 10033 on
  display 0

```

Figure 5 – Extract of the logs generated by an application that performs offline authentication

10. Authentication methods: The purpose of this point is to determine which methods the application uses to allow access. Generally, mobile applications use two authentication methods, to find the corresponding code, the results of point B.7 are scanned in the JD-Gui tool. At this point it should be noted that the JD-GUI search will display results as written in the search box, case sensitive.
11. Blocking: In case of anomalous behavior, for example, if a user enters invalid credentials into the application several times, the application must be blocked. At this point you will find the code that performs this action. To find it, you could enter invalid credentials when you authenticate in the application, and observe the logs generated.
12. Unique authentication: At this point it is checked whether the application performs the authentication process each time it is opened, or whether this process is performed only the first time it is accessed. To determine this, it is used the logs generated by the application when it is opened.

The applications use several methods that control the behavior of the same one, the method which is relevant for this point is the `onResume`. This method is the one that is executed when the application is opened. As shown in Figure 6, this application executes this method when the application is opened. Using the JD-Gui tool, the `onResume` block is searched to find the associated code.

```

1 04-26 22:44:12.260 I/** ( 3406): AplicacionActivity.onResume
2 04-26 22:44:12.260 I/** ( 3406): AplicacionActivity.showRestriction

```

Figure 6 – Extract of the logs obtained when opening the application



For instance, the onResume method has been found as shown in Code 2, this method calls the showRestriction method, which calls the Login screen every time the application opens or returns to the foreground. Therefore, this application requires credentials to be entered each time it is opened.

13. Two step authentication: To allow access, a application must maintain an adequate level of security, one of the ways of achieving this is through two-step authentication (Entersekt, 2014).

```

1  public void onResume()
2  {
3      RUMApplicationHook.onActivityResumed(this);
4      showRestriction();
5      HP_WRAP.onResume();
6      ActivityHooks.onActivityResumeEndHook(this);
7  }
8
9  protected boolean showRestriction()
10 {
11
12     showLoginScreen();
13
14 }

```

Code 2 – onResume method and show Restriction Validate Direct Transfer Parser method

During a two-step authentication, in the first phase the user correctly enters his or her credentials in the application, while in the second phase a verification code is sent to the customer's email or cell phone number, this code must be entered in the mobile application to access.

To identify this process, it is necessary to use the initial exploration of the application, carried out during the information gathering phase, and the logs generated by the application. It is useful to search for words such as code, activation, validate, etc. The words identified in the logs are then searched in the application code.

14. Automated static analysis: Finally, at this point, the .apk file of the application to be analyzed is dragged to the MobSF application web interface. MobSF performs automated analysis of Android applications and this results complement the information about the overall state of application security. Once the analysis is complete, the results are presented and can be saved in.pdf format.

The results obtained are separated into different sections and to export them as a.pdf file, the Download Report option is used and the file is automatically extracted for later analysis.

### 3.3. Dynamic Analysis

This section is called dynamic because it provides insight into the communication aspects. For each of the tasks which are part of the dynamic analysis, it describes how to use the tools, as well as the recommendations for obtaining the desired information. The tasks for this phase are: Communication protocols used, Existing scripts for vulnerability scanning, Content provider, and Data leakage.

1. Communication protocols used: Since applications communicate with a web server for both authentication and transaction processing, it is necessary to verify that this communication is done using secure protocols. Two Network Connections and OWASP ZAP tools are used for this purpose.
2. Existing scripts for vulnerability scanning: There are scripts that help to perform the scanning process, such as Pidcat; or that provide information about vulnerabilities that the application may contain, such as Manintree.

For the development of this point, the Pidcat and Manintree scripts have been considered according to what has been suggested in documents such as: How to Avoid Development Errors in Android (Welivesecurity, 2016) or Risk Analysis of Android Based Appliance (Network Intelligence, 2011), since these scripts have proven to be very useful and have obtained successful results.

3. Content provider: Drozer tool is used to develop this point. A content provider, when not properly configured, shares the information it handles, making it accessible to other tools installed on the same Android device. The procedure for testing this point is as follows.

From the Drozer console, the installed applications are listed with the following command. It uses the name of the application in order to obtain the name of the package, which begins with "com." + the name of the application.

```
>run app.package.list -f application_name
```

Now that the name of the package is known, the following command is executed to know if there are exported content providers. In addition, the application also shows if there are activities, broadcast receivers, or services that are exported.

Within the command syntax, the package\_name is the one obtained in the previous step.

```
>run app.package.attacksurface package_name
```

Next, the information about the permissions that the content provider has is gotten with the following command:

```
>run app.provider.info -a package_name
```

In this way, it is confirmed whether or not the information managed by the content provider is accessible by other applications. For example, if a permission has the value of null, it means that it can be accessed by another application.

4. Data leakage: Finally, the .apk file contains the URIs with which the application communicates and, by means of the Drozer tool, tests are carried out to determine whether or not these addresses can be queried directly. To do this, proceed as follows:

From the Drozer console, the following command is executed to obtain the URIs with which the application communicates. At the same time, Drozer attempts to consult these addresses and reports the results.

```
>run scanner.provider.finduris -a package_name
```

If accessible URIs are found, the following command is used to obtain the information it contains.

```
>run app.provider.query uri
```

Instead, the following command is used to check for SQL Injection vulnerabilities:

```
>run scanner.provider.injection -a package_name
```

With this, it is known whether or not the application contains URIs that are vulnerable to this type of attack.

## **4. Discussion**

The protocol was performed on a mobile banking application of Ecuador. A summary of its results is presented below.

### **4.1. Data Gathering**

The protocol defined for the collection of information has given successful results for this mobile application, since it has been possible to identify the transactions they allow to carry out, as well as the permissions they request for their installation. In addition, it was found at the analyzed mobile application important data, such as the name of the company that developed the application, on its certificate.

### **4.2. Static Analysis**

By applying the protocol defined for static analysis, successful results have been obtained for this mobile application. However, for applications that contain most of the server-side process, such as the analyzed application. For this case, it is recommended to browse through the files which are part of the application.

It has been found that the protocol was most useful for applications which log a large part of the information, so that it is possible to identify the code portions associated with authentication processes, transfers, etc. For the analyzed application, the protocol allowed to identify the folder containing the HTML code files for the different processes to be identified. As an alarming problem it has been found that the banking application, contains the image of a check, including personal information such as, names and valid identification numbers, incorporated in the application code. Also, the analyzed application has been signed using the SHA1 algorithm with RSA, which is known to have collision problems.

### **4.3. Dynamic Analysis**

The protocol has proven to be effective for the dynamic analysis of mobile applications, since the results obtained when applied showed that there were security shortcomings, at the analyzed application. It has been successful, mainly for detecting the use of weak protocols, as it has been possible to identify the username and password for the analyzed application in plain text.

## 5. Conclusions

- Thanks to the selection of specific steps of the methodology of OWASP Mobile Security Project and considering the treats, which affects mobile applications, it's been designed the first version of a testing protocol to verify the security status of mobile applications. The protocol has three phases: data gathering, static analysis and dynamic analysis, which are implemented, using Android Virtual Device Manager, OpenSSL, Dex2Jar, JD-Gui, Logcat, MobSF, OWASP ZAP and Drozer.
- The protocol proposed for the information gathering phase was useful for analyzing the content of the.apk file. The protocol proposed for the static analysis phase has given good results, mainly for applications that perform data processing in the application and not only on the server side. The protocol proposed for the dynamic analysis phase has been successful, mainly for applications that need to authenticate with an external server. For this phase is recommended to use a physical Android device.
- Through the analysis of the chosen mobile application, it's observed that there exists the need of a better control on the developing process of mobile applications, since this application include vulnerabilities such as: customer data included in the source code, and the log of sensitive information. In addition, it's been found that the mobile banking application includes in its source code, personal information of customers at the image of a check, which undermines the privacy of this customer. This highlights the lack of security controls on the developing process. All this can compromise the personal information of the customers.

## References

- Apache Cordova. (n. d.). Mobile apps with HTML, CSS & JS. *Cordova*. Retrieved from <https://cordova.apache.org/>
- Burns, J. (2009). Mobile Application Security on Android: Context on Android security. BlackHat US. *Isec Partners*. 1–27. Retrieved from <http://www.blackhat.com/presentations/bh-usa-09/BURNS/BHUSA09-Burns-AndroidSurgery-PAPER.pdf>
- CVE. (n. d.). Current CVSS Score Distribution For All Vulnerabilities. CVE Details. Retrieved from <https://www.cvedetails.com/>
- Deitel University of London & Trinity College Dublin. (2012). *Android for programmers: an app-driven approach*. Deitel developer series.
- Entersekt (2014). Securing the mobile banking channel. *Entersekt*. Retrieved from <https://www-304.ibm.com/partnerworld/gsd/showimage.do?id=40237>
- Exploit Database. (n. d.). Exploits Database by Offensive Security. *Exploit-db*. Retrieved from <https://www.exploit-db.com/>
- Godfrey, N. (2012). *Decompiling Android*. (M. James, A. Steve, B. Ewan, & C. Corbin, Eds.) (1st ed.). New York: Apress.

- Ionic Framework. (n. d.). One codebase. Any platform. *Ionic Framework*. Retrieved February from <https://ionicframework.com/>
- Network Intelligence (2011). Risk Analysis of Android Based Appliance - Checkmate. *Niiconsulting.com*. Retrieved from <http://niiconsulting.com/checkmate/2011/11/risk-analysis-of-android-based-appliance/>
- NVD-Search and Statistics. (n. d.). Search Vulnerability Database. *Nvd.nist.gov*. Retrieved February 28, 2018, from <https://nvd.nist.gov/vuln/search>
- Owasp.org (n. d.). OWASP Mobile Security Testing Guide. *Owasp.org*. Retrieved from [https://www.owasp.org/index.php/OWASP\\_Mobile\\_Security\\_Testing\\_Guide](https://www.owasp.org/index.php/OWASP_Mobile_Security_Testing_Guide)
- Owasp.org. (2016). Mobile Security Project Archive. *Owasp.org*. Retrieved from [https://www.owasp.org/index.php/Mobile\\_Security\\_Project\\_Archive#tab=M-Security\\_Testing](https://www.owasp.org/index.php/Mobile_Security_Project_Archive#tab=M-Security_Testing)
- Permissions Overview-Android Developers. (n. d.). Permissions overview. *Developer.android.com*. Retrieved from <https://developer.android.com/guide/topics/permissions/overview.html>
- Quora. (n. d.). How do I know if android app was programmed in a framework such as ionic, cordova or phonegap? *Quora*. Retrieved from <https://www.quora.com/How-do-I-know-if-android-app-was-programmed-in-a-framework-such-as-ionic-cordova-or-phonegap>
- Sublime Text. (n. d.). A sophisticated text editor for code, markup and prose. *Sublime Text*. Retrieved from <https://www.sublimetext.com/>
- US-CERT|United States Computer Emergency Readiness Team. (n. d.). Official website of the Department of Homeland Security. *US-CERT*. Retrieved from <https://www.us-cert.gov/>
- Welivesecurity. (2016). Cómo evitar errores de desarrollo en Android con PidCat. *Welivesecurity*. Retrieved from <https://www.welivesecurity.com/las-es/2016/06/28/auditar-aplicaciones-android-pidcat/>
- Zhauniarovich, Y., Philippov, A., Gadyatskaya, O., Crispo, B., & Massacci, F. (n. d.). Towards Black Box Testing of Android Apps. *Zhauniarovich.com* Retrieved from [http://www.zhauniarovich.com/files/pubs/BBoxTester\\_Zhauniarovich2015.pdf](http://www.zhauniarovich.com/files/pubs/BBoxTester_Zhauniarovich2015.pdf)

© 2019. This work is published under  
<https://creativecommons.org/licenses/by-nc-nd/4.0/>(the  
“License”). Notwithstanding the ProQuest Terms and  
Conditions, you may use this content in accordance with the  
terms of the License.