

Secure Android Application Development and Security Assessment

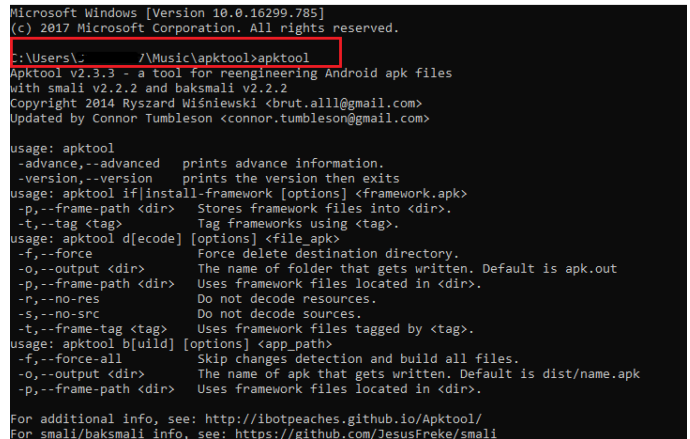
Vandana.H.K¹, Dr. H.D. Phaneendra²

¹Student, MTech- Information Technology, The National Institute of Engineering, Mysuru-09

²Professor & Head, Dept. of CS&E, The National Institute of Engineering, Mysuru-09

Abstract - Nowadays mobile has taken a main part in everyday life of an individual, all important information are stored in mobile database and accessible by mobile applications. Even online transactions are through mobile applications; hence, mobile security requires the main priority. Android applications undergo static and dynamic analysis during security testing. These analysis requires tools like apktool, adb, BurpSuite, Drozer, dex2jar, JDGui and jadx. This paper consists of step-wise procedure for an android application security assessment through which vulnerabilities can be found and solutions to overcome vulnerabilities.

Key Words: BurpSuite, apktool, Drozer, CORS, Verbose Server Banner, Root Detection, AndroidManifest.xml.



```
Microsoft Windows [Version 10.0.16299.785]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\J> cd /Music/apktool
apktool v2.3.3 - a tool for reengineering Android apk files
with smali v2.2.2 and baksmali v2.2.2
Copyright 2014 Ryszard Wiśniewski <brut.all@gmail.com>
Updated by Connor Tumbleson <connor.tumbleson@gmail.com>

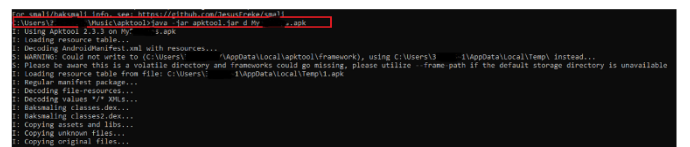
Usage: apktool
  -advance,--advanced prints advance information.
  -version,--version prints the version then exits
Usage: apktool if|install-framework [options] <framework.apk>
  -p,--frame-path <dir> Stores framework files into <dir>.
  -t,--tag <tag> Tag frameworks using <tag>.
Usage: apktool d[decode] [options] <file_apk>
  -f,--force Force delete destination directory.
  -o,--output <dir> The name of folder that gets written. Default is apk.out
  -p,--frame-path <dir> Uses framework files located in <dir>.
  -r,--no-res Do not decode resources.
  -s,--no-src Do not decode sources.
  -t,--frame-tag <tag> Uses framework files tagged by <tag>.
Usage: apktool b[uild] [options] <app_path>
  -f,--force-all Skip changes detection and build all files.
  -o,--output <dir> The name of apk that gets written. Default is dist/name.apk
  -p,--frame-path <dir> Uses framework files located in <dir>.

For additional info, see: http://ibotpeaches.github.io/Apktool/
For smali/baksmali info, see: https://github.com/JesusFreke/smali
```

Fig -1: Result of >apktool; command

1. INTRODUCTION

Mobile plays an important part in recent years and replaced personal computer to more extent. Applications are built for every need including e-commerce, online transactions, health care, etc. These applications consists of sensitive data of an individual and needs to be secured. Only securing android device is not enough one should also look into that each application is secured or not.



```
C:\Users\J> cd /Music/apktool
apktool v2.3.3 on x64
I: Loading resource table... done
I: Decoding android/manifest.xml with resources...
I: WARNING: Could not write to (C:\Users\J\AppData\Local\apktool\framework), using C:\Users\J\AppData\Local\Temp instead...
I: Please be aware this is a writeis directory and frameworks could go missing, please utilize --frame-path if the default storage directory is unavailable
I: Loading resource table from file: C:\Users\J\AppData\Local\Temp\1.apk
I: Reading manifest package...
I: Decoding file-resources...
I: Decoding values *?* XML...
I: Baksmaling classes.dex...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
```

Fig-2: Result of >java -jar apktool.jar d -s application_name.apk; command

Android application undergo static and dynamic analysis during security assessment. Generally, static analysis of an android application consists of source code review, side channel leakage, reverse engineering etc. Dynamic analysis consists of review of interception of api between application and server. Static analysis requires apktool, dex2jar, jadx, JDGUI tools. Dynamic analysis requires adb, Drozer, BurpSuite tools.

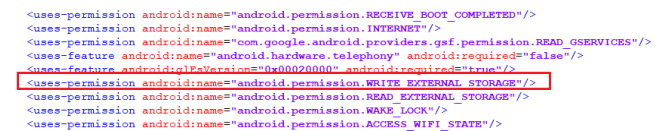
2. METHODS

2.1 Method-I

Once an android application is given for security assessment, it will be in .apk form. Apktool is the tool utilized for reverse engineering of an android application where we can decompile full source code into .smali form. Even modifying and recompiling .apk is easy using this tool. Download the apktool for free from google as it is an open source. Once installed, using following command in terminal you can reverse engineer the .apk file.

2.1.1 Problem A

Consider an example for Dangerous permissions used vulnerability, as “write_external_storage” permission set as shown in figure 3. This permission allows the application to store the application data onto the external storage like memory card, which is accessible to all other applications. Anything stored in the external storage is accessible to all the application irrespective the device is rooted or not. Confidentiality of the sensitive data is compromised, as the data is openly accessible.



```
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="com.google.android.providers.gsf.permission.READ_GSERVICES"/>
<uses-feature android:name="android.hardware.telephony" android:required="false"/>
<uses-feature android:glEsVersion="0x00020000" android:required="true"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
```

Fig -3: AndroidManifest.xml where write_external_storage permission is set.

Solution A

It is recommended to remove this permission and allow only those permissions, which are needed or required for the

application to work properly. [6] Provide knowledge about all uses-permission in android.

2.1.2 Problem B

Consider debuggable flag enabled vulnerability as shown in fig- 4, the debug tag in AndroidManifest.xml defines whether the application can be debugged or not. If the application can be debugged then it can provide plenty of information to an attacker since the local storage of the application is accessible to attacker and it will also make easy in gaining intercepted API between application and server. Attacker needs to have physical access to rooted devices or there should be a malware app running in background, which can read through unencrypted sensitive data saved within sandbox of the app. Proxy can be adjusted and interception can be done using Burp Suite, this will explained in next method.

```
<?xml version="1.0" encoding="UTF-8"?>
<manifest package="com.nesteffil.loginmethodhooking" xmlns:android="http://schemas.android.com/apk/res/android"
  application android:theme="@style/AppTheme" android:supportsRtl="true" android:label="@string/app_name" android:icon="@mipmap/ic_launcher" android:debuggable="true" android:allowBackup="true">
  <activity android:name="com.nesteffil.loginmethodhooking.LoginActivity">
```

Fig -4: AndroidManifest.xml where debuggable and allowBackup flags are set to true

Consider allowBackup flag enabled vulnerability as shown in figure 4, this flag allows anyone to backup your application data via adb. It allows users who have enabled USB debugging to copy application data off of the device. Android backups rely on the Android Debug Bridge (ADB) command to perform backup and restore. ADB, however, has been a soft target for attacker and is still not trusted by respected developers. The idea that someone can inject malicious code into your backup data is unsettling. Attackers can read its internal sensitive data like password, tokens etc. and modify these sensitive data or configurations.

Even though the flag is set to false or flag is only removed, attacker can modify them and recompile .apk file ,as shown in following steps

Step 1: Open AndroidManifest.xml file in text editor and add android:debuggable="true" inside <application> tag as shown in fig- 4.

Step 2: Recompile the above modified files using following command.

```
>java -jar apktool b modified_folder -o modified_app_name.apk
```

Step 3: The new .apk formed should be signed by apk signer application before installing into mobile.

Solution B

It is recommended that Android applications that are not in the production state are expected to have this attribute set to true to assist the testers and developers however before

the actual release of the application this tag should be set to false. It is also recommended to Set android:allowBackup=false in Androidmanifest.xml or even better is to remove that tag.

```
<?xml version="1.0" encoding="UTF-8"?>
<manifest package="com.nesteffil.loginmethodhooking" xmlns:android="http://schemas.android.com/apk/res/android"
  application android:allowBackup="false" android:debuggable="false" android:icon="@mipmap/ic_launcher" android:label="@string/app_name" android:largeHeap="true" android:usesCleartextTraffic="true">
```

Fig -5: AndroidManifest.xml where debuggable and allowBackup flags are set to false

2.1.3 Problem C

Consider Improper export of android application components vulnerability as shown in figure 5, The Android application exports a component for use by other applications, but does not properly restrict which applications can launch the component or access the data it contains. Intents can be used to launch an activity, to send it to any interested broadcast receiver components, and to communicate with a background service. Intents messages should be reviewed to ensure that they does not contain any sensitive information that could be intercepted. If access to an exported Activity is not restricted, any application will be able to launch the activity. This may allow a malicious application to gain access to sensitive information, modify the internal state of the application, or trick a user into interacting with the victim application while believing they are still interacting with the malicious application. If access to an exported Service is not restricted, any application may start and bind to the Service. Depending on the exposed functionality, this may allow a malicious application to perform unauthorized actions, gain access to sensitive information, or corrupt the internal state of the application.

```
dz> run app.package.attacksurface com.isi.testapp
Attack Surface:
  2 activities exported
  0 broadcast receivers exported
  0 content providers exported
  0 services exported
  is debuggable
dz> █
```

```
dz> run app.activity.info -a com.isi.testapp
Package: com.isi.testapp
  com.isi.testapp.MainActivity
  com.isi.testapp.Welcome
dz> █
```

Fig -6: Drozer utility used to list activities exported in the target application.

MWR Labs developed a framework called Drozer, which is one of the best Android security assessment tools available for security assessments. To perform a security assessment using Drozer, the users will run the commands on console of their workstation and it is sent to an agent who execute the task.

com.isi.testapp.MainActivity is the home screen which in order to be launched, it should be exported. Here, **com.isi.testapp.Welcome** is the name of the activity which is triggered after the login screen. Using Drozer, launch that activity.

```
dz> run app.activity.start --component com.isi.testapp com.isi.testapp.Welcome
dz> □
```

Fig -7: Drozer utility used to exploit vulnerabilities

This command provides an appropriate intent in the background in order to launch the activity, which is to bypass authentication during logging in to an application.

Solution C

It is recommended to make `android:exported=false` for unwanted functionalities in `AndroidManifest.xml`.

```
<activity android:exported="false" android:name="com.google.android.gms.appinvite.PreviewActivity" android:theme="@style/Theme.AppInvite.Preview" >
    <intent-filter>
        <action android:name="com.google.android.gms.appinvite.ACTION_PREVIEW"/>
        <category android:name="android.intent.category.DEFAULT"/>
    </intent-filter>
</activity>
```

Fig -8: AndroidManifest.xml where exported flag set to false

2.2 Method-II

After all static analysis, let's see how BurpSuite will be used for security assessment. Download the BurpSuite-Community Edition for free from google as it is an open source. Set every settings required to intercept the traffic, exchanged between application and server. Proxy settings is similar to what we set during web application pentesting, but the change is instead of localhost we choose all interface option. In mobile device, we have to set proxy according to machine IP address, which has burpsuite running on it.

Here we discuss some basic vulnerabilities, which are found during this assessment.

2.2.1 Problem D

Consider, Cross Origin Resource Sharing(CORS) vulnerability where CORS defines whether resources on other domains can interact with this server. An attacker can place malicious JavaScript on his domain that can exploit the unrestrictive CORS policy to access sensitive data on this server or perform sensitive operations without the user's knowledge. Additionally, an attacker could exploit security vulnerabilities on other domains to compromise services on this server. The CORS policy relaxes the Same Origin Policy, an important security control that isolates potentially malicious resources to its respective domain name. If a script attempts to violate the Same Origin Policy by interacting with another domain, modern browsers will check a server's CORS policy by issuing a "pre-flight request". The browser allows the interaction only if the server responds with an Access-Control-Allow-Origin header that

lists the script's domain or a wildcard match (*). A wildcard match allows interaction from any other domain, which allows any malicious content to retrieve content from this server or perform user actions.

An unrestricted CORS policy allows an attacker to access sensitive data or perform unauthorized user actions without user knowledge. Malicious JavaScript can perform these actions even if the server uses Cross Site Request Forgery tokens. Hence, an attacker can access sensitive data of victim. This vulnerability comes under category M3-Insecure Communication.

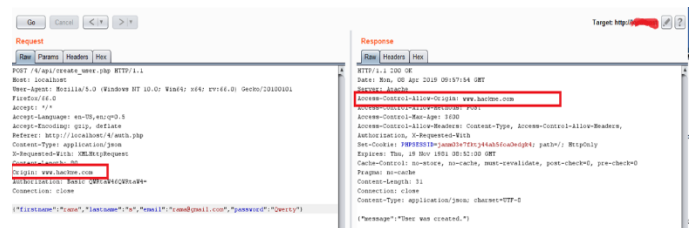


Fig -9: Access-Control-Allow-Origin header in response reflecting injected Origin in request

Solution D

The Access-Control-Allow-Origin header should not be set to a wildcard match. In most cases, this header can be safely removed. However, if the application requires a relaxation of the Same Origin Policy, the Access-Control-Allow-Origin header should whitelist only domains that are trusted by this server.

```
header("Access-Control-Allow-Origin: http://localhost/4/api/");
header("Content-Type: application/json; charset=UTF-8");
header("Access-Control-Allow-Methods: POST");
header("Access-Control-Max-Age: 3600");
header("Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers, Authorization, X-Requested-With");
```

Fig -10: Whitelisting trusted domains for server

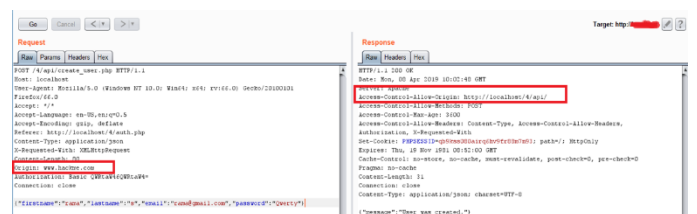
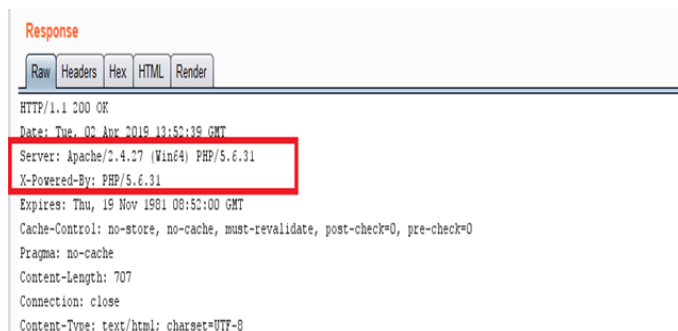


Fig -11: Access-Control-Allow-Origin header in response reflecting only trusted domains.

2.2.2 Problem E

Consider, Verbose Server Banner vulnerability where the server information is sent in the HTTP responses from the server. The information is commonly included in the server response headers and can disclose information like server name, type, and version number. By knowing version, type of webserver, how each type of web server responds to specific commands and keeping this information in a web server fingerprint database, an attacker can send these commands

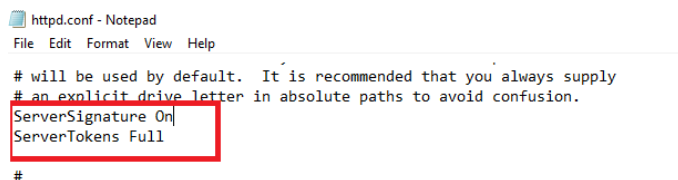
to the web server, analyze the response, and compare it to the database of known signatures. By knowing the information about the server, an attacker can plan attacks in future, with the information obtained. There may be publically known exploits and vulnerabilities associated with the server hosted, whose information gets disclosed in the banner. Verbose server banners provide additional information that allows an attacker to perform targeted attacks to the specific technology stack in use by the application and underlying infrastructure. This vulnerability comes under category W6 – Security Misconfiguration.



```

Response
Raw Headers Hex HTML Render
HTTP/1.1 200 OK
Date: Thu, 02 Apr 2019 13:52:18 GMT
Server: Apache/2.4.27 (Win64) PHP/5.6.31
X-Powered-By: PHP/5.6.31
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Content-Length: 707
Connection: close
Content-Type: text/html; charset=UTF-8
    
```

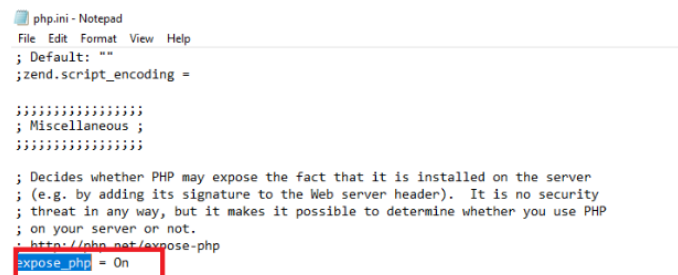
Fig -12: Server banner disclosure (not secured)



```

httpd.conf - Notepad
File Edit Format View Help
# will be used by default. It is recommended that you always supply
# an explicit drive letter in absolute paths to avoid confusion.
ServerSignature On
ServerTokens Full
#
    
```

Fig -13: Server Apache has above configurations in it's httpd.conf file



```

php.ini - Notepad
File Edit Format View Help
; Default: ""
;zend.script_encoding =

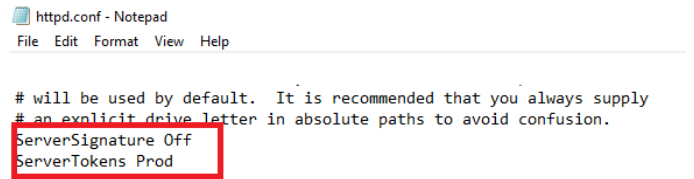
; Miscellaneous ;

; Decides whether PHP may expose the fact that it is installed on the server
; (e.g. by adding its signature to the Web server header). It is no security
; threat in any way, but it makes it possible to determine whether you use PHP
; on your server or not.
; http://php.net/expose-php
expose_php = On
    
```

Fig -14: PHP has above configurations in its php.ini file

Solution E

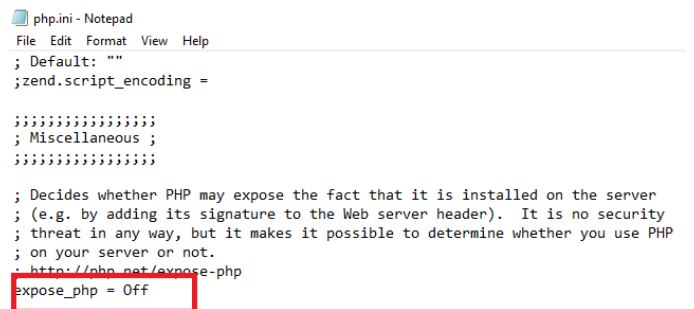
Verbose server information should be removed from all HTTP responses. This can be performed by modifying the server's configuration files. It is recommended to use generic error message response from server, so that server banner is not disclosed in the error message response from the server.



```

httpd.conf - Notepad
File Edit Format View Help
# will be used by default. It is recommended that you always supply
# an explicit drive letter in absolute paths to avoid confusion.
ServerSignature Off
ServerTokens Prod
    
```

Fig -15: Changing configurations of Server Apache in its httpd.conf file



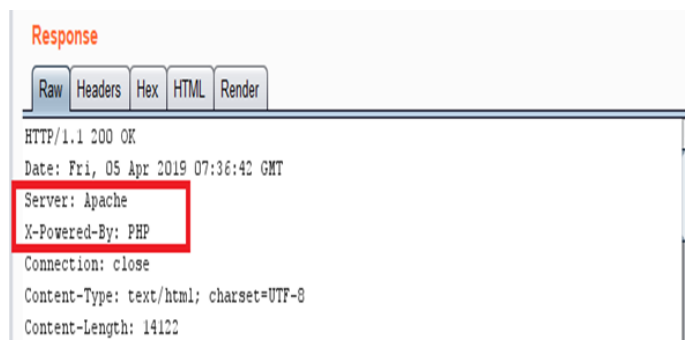
```

php.ini - Notepad
File Edit Format View Help
; Default: ""
;zend.script_encoding =

; Miscellaneous ;

; Decides whether PHP may expose the fact that it is installed on the server
; (e.g. by adding its signature to the Web server header). It is no security
; threat in any way, but it makes it possible to determine whether you use PHP
; on your server or not.
; http://php.net/expose-php
expose_php = Off
    
```

Fig -16: Changing configurations of PHP in its php.ini file



```

Response
Raw Headers Hex HTML Render
HTTP/1.1 200 OK
Date: Fri, 05 Apr 2019 07:36:42 GMT
Server: Apache
X-Powered-By: PHP
Connection: close
Content-Type: text/html; charset=UTF-8
Content-Length: 14122
    
```

Fig -17: Server banner after changing configurations of servers.

2.2.3 Problem F

Consider, Improper Cookie Configuration vulnerability where the session cookie is set without including the HttpOnly and Secure flags. The HttpOnly flag helps mitigate (but doesn't completely prevent) the risk of client side scripting accessing the cookie value (assuming the browser supports the flag setting). This helps prevent cross-site scripting from accessing the cookie from the DOM, leading to potential session hijacking. The secure flag ensures that the cookie is not transmitted in an http request, forcing the cookie to be transmitted only when using HTTPS. This helps prevent disclosure of the session cookie if a request is submitted using HTTP instead of HTTPS. Cookie value will be transmitted unencrypted over http which leads malicious or unintentional disclosure of the session cookie value. Attacker can use this vulnerability to perform cross site scripting and session hijacking.



Fig -18: Cookie with no httponly or secure flags

Solution F

The secure flag and Http only flag should be set on all cookies that are used for transmitting sensitive data when accessing content over HTTPS.

```
<?php
session_start();
$currentCookieParams = session_get_cookie_params();
$sidvalue = session_id();
setcookie(
    'PHPSESSID',//name
    $sidvalue,//value
    0,//expires at end of session
    $currentCookieParams['path'],//path
    $currentCookieParams['domain'],//domain
    true,//secure
    true //HttpOnly
);
```

Fig -19: Updating setcookie with httponly or secure flags in code



Fig -20: Cookie with httponly or secure flags.

2.2.4 Problem G

Consider, lack of root detection vulnerability where the application will not detect rooted android device. Android implements containerization so that each app is restricted to its own sandbox. A regular app cannot access files outside its dedicated data directories, and access to system APIs is restricted via app privileges. As a result, an app’s sensitive data as well as the integrity of the OS is guaranteed under normal conditions. However, when an adversary gains root access to the mobile operating system, the default protections can be bypassed completely. Attacker should have physical access on the device, however rooted device remotely accessible in case user root SSH not protected. Applications on a rooted device run as root outside of the android sandbox. This can allow applications to access sensitive data contained in other apps or install malicious software negating sandboxing functionality. The risk of malicious code running as root is higher on rooted devices, as many of the default integrity checks are disabled.

Solution G

Developers of apps that handle highly sensitive data should therefore consider implementing checks that either prevent the app from running under these conditions, or at least warn the user about the increased risks.

3. CONCLUSIONS

This paper provides basic vulnerabilities found during security assessment and solutions for developers to overcome them. The main aim of this paper is to provide basic improvements that can be done during an android application development so that during security assessments, tester can concentrate on major vulnerabilities rather than spending time on testing these basic vulnerabilities.

ACKNOWLEDGEMENT

I would like to thank Dr. H.D. Phaneendra, Professor & Head, Dept. of CS&E at NIE, Mysore for his guidance and support in doing case study of the above paper.

REFERENCES

Karthick S, Dr. Sumitra Binu- “Android Security Issues and Solutions”- International Conference on Innovative Mechanisms for Industry Applications (ICIMIA 2017).

- [1] Igor Khokhlov, Leon Reznik -“Android System Security Evaluation”- 2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC).
- [2] Sebastian Lekies, Martin Johns and Walter Tighzert – “The State of the Cross-domain Nation”- 2011 IEEE Security.
- [3] <https://www.medianova.com/en-blog/2019/02/20/a-technical-introduction-to-cors-cross-origin-resource-sharing>. [Online]
- [4] <https://technumero.com/disable-server-signature-by-editing-htaccess-apache/>. [Online].
- [5] Cheng-Liang Kuo , Chung-Huang Yang-“Security Design for Configuration Management of Android Devices”- 2015 IEEE 39th Annual Computer Software and Applications Conference.