NSysS 2017.
Department of CSE, BUET. 5-8 January, 2017

Vulnerability Detection in Recent Android Apps: An Empirical Study

Faysal Hossain Shezan[1], Syeda Farzia Afroze[2], Anindya Iqbal [3]
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology
[1]faysalhossain2007@gmail.com, [2]farziaa@gmail.com,[3]anindya@cse.buet.ac.bd

***Abstract*—With the continuous and rapid increase in quantity and diversity of Smartphone application usage, the storage of sensitive personal and even financial information of the users is also being augmented. It creates motivation for developers of malicious applications to put more effort on discovering ways to identify and exploit the vulnerabilities of utility applications and grab the sensitive information of the users. Android applications, being more open in nature and popular among armature individual developers, fall victim to the malwares quite frequently. Recently, the Govt. of Bangladesh has taken initiative to encourage and patronize young developers to develop utility apps for free public use in the context of Bangladesh (app source: EATL[1]). While the motivation is great, i.e., benefiting common people, the way these are developed and released have reasons to suspect that recent vulnerabilities may exist due. This may harm the users and ruin the good initiative. In this paper, we have carried out an empirical study on a selected set of these apps to detect eight common vulnerabilities. We have carefully chosen three quality tools that cover testing of all these vulnerabilities. We reported the detected results showing vulnerabilities in the tested apps, presented statistics of the vulnerabilities and discussed countermeasures. We believe this study would benefit the developers and indirectly the potential users of these applications.**

***Index Terms*—Android Malware, Static Analysis, Dynamic Analysis, Vulnerability, App security, Security testing.**

I. Introduction

In modern days, with the rise of ubiquitous use of smartphones, the usage of different utility applications is growing rapidly. Mobile devices are taking place of traditional PCs by bringing convenience to people that was unthinkable even a decade ago. The number of smart-phone users all over the world was 1.5 billion in 2014 and is forecast to grow to around 2.5 billion in 2019 [1]. Over 36% of the world's population is projected to use a smart-phone by 2018, up from about 10% percent in 2011. Android and iOS are the two most popular Smartphone operating systems in the industry. Android, with 80% of market share has topped the list [1]. As of February 2016, Google Play Store contained 2 million apps [2].The software designed to damage or do other unwanted actions on a system are termed as malware. According to Lookout [3], there is a 75% year-over-year increase in U.S. mobile malware rates in 2014.

[1]http://www.eatlapps.com/

978-1-5090-3260-0/17/$31.00

A beauty of mobile app development, especially in open platform like Android, is that young developers with interesting ideas may form small teams and develop apps and release among a vast audience. The vulnerabilities cause from lack of awareness of developers or inherent weakness of platform used. These developers may not be cautious about all the security risks and the practices to follow to avoid these. As a result, these apps may become easy targets for malicious sources. There are an enormous number of applications in Play Store itself. These apps are not comprehensively tested for malicious behavior or vulnerability when they are published, making it an attractive target for the attackers [4]. There are also third-party app-stores that barely perform any testing while publishing apps. A user can thus end up with an app vulnerable to malicious activities even after downloading it from an apparently reliable source. Here lies the importance of research on classification of these vulnerabilities, detecting and providing countermeasure.

Due to the rise of smart phones and their apps, Government of many countries are coming forward to develop applications promoting health , travel, education, consumer price sharing, information sharing etc. aimed for the usage by mass people. To make the young generation interested in app development and IT, Bangladesh Government has taken several campaigns, workshops and competitions to develop such apps. In such initiatives, the amateur or inexperienced developers are developing apps with innovative ideas. However, such development process hardly has opportunity to be tested by security experts, unlike the scenario in established software development companies. Again, these applications are aimed at mass people of the country, who are also not conscious enough about the security of their data. This results in an attractive entry point for attackers to exploit; causing theft of private personal data or government information. Our work is directed at finding the vulnerabilities of recent Android apps and to find their remedies, wherever possible.

We have noted some of the common vulnerabilities from the recent literature. A number of works have found to be on detecting SMS Malware attacks and preventing them. According to a Survey by Kaspersky [5], 33.5% of all attacks were Trojan-SMS attacks, resulting in unexpected charges on user's phone bill or being signed up for unexpected premium services. They also point out that the older versions of Android (versions 4.1.x and older) are at risk of being exploited by

scripts the download and execute malwares. These are known issues and fixed in later versions of android. Hence, the apps that support outdated Android versions are at the risk of experiencing these attacks. Component Hijacking is another vulnerability of recent android apps as identified in [6] . A large number of work are also found to attack vulnerabilities regarding Web-view in Android [7], [8] . Leem et. al. [9] has proposed a simplified single process to detect most common vulnerabilities in Android apps in his paper.

In this paper, we first discussed the causes and effects of relevant weaknesses of Android apps, along with the ways to eliminate them. Afterwards, we have chosen a number of applications from the EATL app store and tested them with a few well known testing tools. EATL app store is a third party app store, aimed at promoting apps developed by Bangladeshi developers. We have also checked vulnerabilities in some popular Bangladeshi apps referred by Audacity[2]. We have discovered that all of the chosen apps are vulnerable to one or more threats that we tested.

We have chosen twenty nine apps from local app stores to discover their vulnerabilities. We assumed these apps to be more vulnerable because they are published in a local store. These apps are free for use, and aimed at the mass people of the country. The local app stores usually execute minimal security tests while publishing apps, letting vulnerable or even malicious apps to slip in sometimes. The chosen apps are mostly from individual developers. Furthermore, we have tried to maintain a wide range of diversity among the apps. Some of the apps interact directly with the users, while some show static data from either a remote server or local database. Some support Webview, while others simply make use of activities and intents to pass data. We have also tested 6 apps randomly chosen from the most popular apps from the Google Playstore.

We have chosen three tools for testing the applications for vulnerabilities. The reason for choosing them is that these three tools cover extensively all the vulnerabilities that we discuss in this paper. Two of these tools are open-source, leaving out scope for future extension by us in case of detecting other vulnerabilities. They are also popular tools, free to use, and available for most operating systems.

The remainder of this paper is organized as follows.
Section II analyzes the research works that have been done so far in this field. We have discussed the current form of app vulnerabilities and made some classifications in section III. We have picked three different tools based on certain characteristics which is described in section IV. Finally the results are given in section V, and section VI concludes the paper.

II. Related Work

According to OWASP (Open Web Application Security Project), vulnerability is just like a hole in application that allows attacker to cause damage to stack holders of an application. It can be a design flaw or implementation bug. Once an attacker is able to find a vulnerability in an application and determines how to access it, the attacker is likely to exploit the vulnerability. These kinds of crimes mainly target the two basic principles of app security which is *Confidentiality* and *Integrity* of the resources. Around 90% of all vulnerabilities are contained in application layer [10]. Here, we discuss some [11]–[31] recent works that aim to detect the vulnerabilities.

A. Vulnerability Detection Techniques

Since security threats are increasing and cause serious damage to the users, researchers are also working in numbers to detect and prevent the malicious apps [11]–[16]. Detection tools generate report and then applies Machine Learning anomaly detectors to classify the collected data [17], [18]. Faruki et al. discussed the Android security enforcement mechanisms and threats to the existing security enforcements [19]. All the malware growth timeline between 2010 and 2014 are noted in this paper. Based on the analysis of available tools he proposed a Hybrid Approach for analyzing android malware. Vidas et al. [20] has explored Android security features to discover loopholes and vulnerabilities. He has built taxonomy of attacks on the Android OS. Based on this, he has also suggested on the security properties modern mobile operating systems like Android should possess.

Liang et al. [21] has categorized the threats and delineate those.He divided the existing works into privacy protection enhancement and privacy leakage detection sections.

B. Comparison of Static and Dynamic Analysis

Security vulnerability issues caused codes interaction with other system components like SQL databases, Web services or application servers are detected by Dynamic analysis. Combining both analysis should cover 95%of the flaws.

Chex et al. built a static analyzer, Dalysis which performs on android bytecode [22]. Based on that framework and considering current app vulnerabilities they designed a static analyzer CHEX. It automatically detects multiple entry points in Android app in an efficient and accurate way. Yang et al. [23] presents an app validation framework named App Intent that extracts app inputs that represent user interactions in an acceptable amount of time to determine if data transmission from an app is indeed intended by the user. Schmer et al. [24] describes an approach using a combination of static analysis and run-time management, based on software architecture models, named Raindroid, that can improve security along with maintaining framework extendibility. It identifies potentially vulnerable communication patterns, while adapting the system to deny, allow or request permission from the user. Several of the recent works feature the use of machine learning on detecting vulnerabilities [25]–[31]. Yuan et al. proposed Droid-Sec [26], an ML-based method making use of more than 200 features extracted from static analysis and dynamic analysis of apps. Draco , proposed by Bhandari et al. [25] is a two-phase learning system, also blending static and dynamic analysis. Canfora et al. suggested a malware detection method

[2]http://audacityit.com/android/bd-android-apps/

in [27] by applying learning techniques on specific system-call sequences.

III. Discussion on App Vulnerabilities

In many cases attackers may get information from user device through the app. There are some features that need to be protected. App vulnerability checker software searches for such privileges that are permitted in the app.

1) **Vulnerability in Storage Access**

Android provides several options to store data from the apps locally such as Shared Preferences, SQLite Database, External storage, and Internal storage.

Files created by an app in internal storage are accessible only to that app. Android has implemented this protection of data storage which works sufficiently in most cases. However, sometimes developers use MODE_WORLD_READBALE and MODE_WORLD_WRITABLE so that some other particular app can access them. This renders the files accessible to malicious apps too; which might result in security breach. A solution to this can be avoiding MODE_WORLD_READBALE and MODE_WORLD_WRITABLE or hardcoded private data such as encryption or decryption keys, and adding an extra layer of encryption.

Some apps write files to the external storage. Files created on external storage are readable and writable. Thus it is imprudent to save when the files contain important and private data. All the important files and database file should be stored in internal storage. Then attacker will find it difficult to get access to those important files. While reading data from external storage, it is advisable to perform input validation. Executables or class files should not be saved in or retrieved from external storage, and if such a need arises, these executable should be encrypted and cryptographically verified. Even the Shared Preferences and local SQLite databases become accessible when the device is rooted. Therefore, in case of theft of phone, sensitive data stored in these data storage may fall into wrong hands.

Data may be leaked from activities and intents while they are saved as extras. These extras can be read from anywhere unless an additional permission is added. If private keys used by an app are saved in source code, with no or minimal security measures taken, it can be easily stolen or reverse engineered by malicious parties.

2) **Attack in Webview**

Android Webviews are not vulnerable by themselves. For instance, Gmail app uses Webview to show emails safely. The vulnerability of Webview depends on the content that is shown in them. Viewing arbitrary third-party content in Webview of an app may turn out to be extremely harmful. In cases like these, browsers usually sandbox the unreliable contents in a separate process. However, since Webview is a single-process, it cannot do so. It, therefore, grants access to malicious contents, giving them same privilege that the app enjoys. Thus, it is absolutely necessary to make sure that no third-party content is being viewed from the Webview. In the cases where allowing user-provided content is necessary, only plain-text should be accepted, after sanitizing and validating it.

Some key vulnerabilities of using Webviews are SQL Injection, Cross-Site Scripting (XSS) and Insecure Direct Object References [32], [33] . All of these are potentially harmful attacks. An attacker can gain access to local private files like shared preferences or send unwanted SMS from the phone, in addition to stealing credentials. Fig. 1 shows one such attack. These attacks can be prevented by setting the setJavaScriptEnabled attribute to false. If that is not possible, each context needs to be escaped properly by using an XSS filter component.

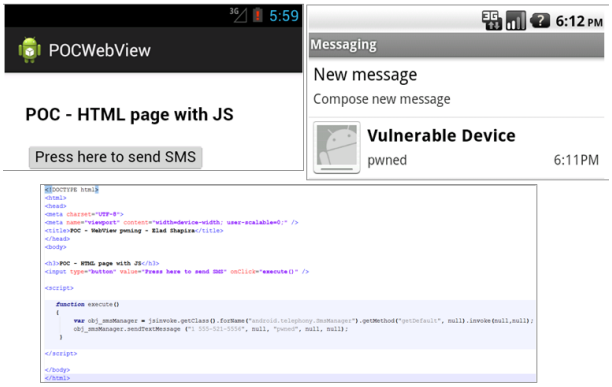

Fig. 1. An example of exploiting Webview to send SMS messages [34]

When setAllowFileAccess attribute of a Webview is set to be true, malicious attackers can inject harmful script in it and exploit the chance of accessing local storage. This can be avoided by setting the aforementioned attribute is false while viewing third party content in Webview.

Last year, Google declared to end Webview extension security support on Android versions Jelly Bean (4.1 - 4.3) and below [35]. This renders the apps supporting the mentioned Android versions prone to Webview attacks. Therefore it is obligatory for the developer to target the most recent API level allowable for an app.

3) **Android SQLite Databases Encryption**

One of the most common places to store huge chunks of local data in android is the SQLite database. It has the ability to roll-back its actions in case of app-crash or device crash. Even after these facilities, it is important to remember that SQLite database stores its contents in a usual file that can be infected by an attacker with garbage values, or data can even be stolen. This may happen when USB debugging is enabled or the device is rooted. With debug mode enabled, SQLite data can be easily retrieved either with ADB, or with DDMS support of eclipse. A demonstration of such an exploit is given in [36]. This vulnerability can cause identity

theft, damage to reputation, fraudulent activities using attained data, external Policy Violation or financial loss. A remedy to this is to add an extra layer of encryption over the usual operative system provided encryption to the database files. SQLite Encryption Extension (SEE), an extension provided by SQLite Database, may be of help in this aspect.

4) **Checking Vulnerability of Intents**

When an android app registers to receive implicit intents from the Android OS, it is registered in such a way that it receives any explicit intent too. In fact, Intent spoofing vulnerability in Android apps is one of the most common issues. A malicious app can send an explicit intent to the target apps while the target app has no way of knowing where the Intent came from. It will treat the malicious app as a trusted one, thus providing a break point to the harmful intent.

An application can be registered to receive broadcasts occurred by system events such as BOOT COMPLETE, SMS RECEIVED, BATTERY LOW, or it can also generate custom broadcast intents for which a receiver can be registered. If the developer doesnt impose some control on who can broadcast and who cannot, harmful apps from untrustworthy sources may be able to spoof broadcasts that are received by the app, causing serious risk. [37] shows an example of exploiting broadcast intents.

Sticky Intents linger in system for future broadcast listeners. Sticky intents stay alive even when the application is reinstalled and doesn't get remove until the device is restarted. If this is not used carefully by the developer, it might cause security risk.

5) **Analyzing Advertisement Module**

Third-party advertising tools, also known as ad-tech, help companies leverage their intent data, which forms the foundation of conversion.

There are three types of digital data: search, interest and intent. Intent data is collected by e-commerce and price comparison companies studying what happens on their sites. A malicious ad-tech may be able to steal data this way if the developer is not careful.

Some malwares function by repackaging legitimate apps from the Google Play store with harmful adware, and then releasing them to a third-party store. Since after repackaging, the app remains fully functional, these malwares become hard to detect. Now, when the user downloads this app from the third party store, the hacker can easily access his sensitive data.

6) **Checking for apps supporting outdated API versions and Sensitive API**

According to Duo Security [38], More than 90% of Android devices are running out-dated versions (4.0 and below) of the operating system. Since Google doesn't provide support for many features of these outdated API versions, these devices become an appealing prey for the hackers. Thus, adding backward compatibility for older API-versions that are not supported by Android can be a threat to the application. It might allow a developer to make use of deprecated packages that may have catastrophic result for security of the app.

7) **Monitoring SMS and Phone call**

SMS malwares are one of the most common types of malwares of recent days [39]. In malicious exploits like Stagefright, discovered in 2015, a hacker creates a short video, hiding the malware inside it. He then texts it to the phone numbers of his prey in form of MMS. As soon as it's received by the target, it completes initial processing, triggering the vulnerability. When this is successful, attacker can have full control over the device, from copying sensitive data to taking control of camera and camera and microphone of the device.

Sometimes malicious third party scripts may trigger unwanted messages or calls transferring sensitive data or adding a premium service, causing a rise in phone bill. Harmful apps from unreliable sources may also modify sms and call settings.

8) **Attack in Android Debug Mode**

Android supports a BuildConfig.DEBUG attribute from in manifest file from SDK Tools version 17 to enable debugging the app when connected to a computer during the development phase. Debug mode allows a developer to transfer data both ways between a computer and device, read log cat data easily, debug application, rapid installation and uninstallation of apps, and accessing a stripped-down shell on the device for command-line interaction. However, a released app should have this attribute turned off. Otherwise, when plugged into an untrusted computer, all the private data becomes available to the attacker.

IV. Discussion on Tools Used

There are lots of tools available for detecting malware and vulnerabilities in android app. Their main purposes are:

- Testing for overall app security
- Assessment and analysis
- Data leakage detection

App Security assessment solutions determine the vulnerabilities, which if exploited, can be harmful to the user and device security. Analysis based solutions detect the malicious behavior within the apps, and the detection solutions aim to prevent the on-device installation. To achieve these steps, static analysis or dynamic analysis can be performed.

There are mainly two type of analysis done by each tools, i.e. Static Analysis and Dynamic Analysis. In static analysis, the app is first installed in the emulator and then it extracts the app data. Then it analyzes the data and generates the first report of the app. Dynamic analysis process starts as soon as the static analysis ends. Dynamic analysis process includes decompiling apk and parse decompiled files. It then analyzes the data and generate the second report. When the static and dynamic reports are generated, it merges them in a single report and show the full analysis report on a particular tested app.

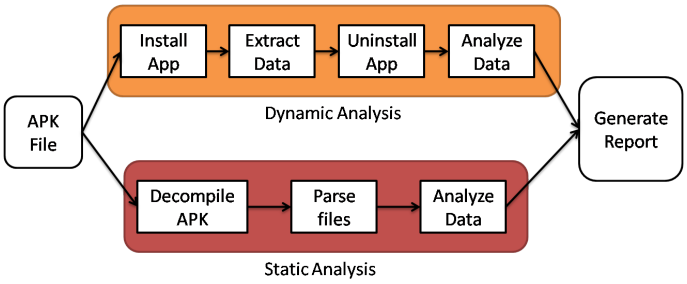

Fig. 2. Working Process of Vulnerability Detection Tools

TABLE I
TESTS SUPPORTED BY TOOLS

Vulnerability	AndroBugs	SandDroid	Qark
Vulnerability in Storage Access	✓	✓	✓
Attack in Webview	✓	×	✓
Android SQLite Databases Encryption	✓	×	×
Checking Vulnerability of Intents	×	×	✓
Analyzing Advertisement Module	×	✓	×
Checking,for,apps,supporting, outdated,API,versions and Sensitive API	✓	✓	✓
Monitoring SMS and Phone call	✓	✓	×
Attack in Android Debug Mode	✓	×	✓

We have used three tools and frameworks to detect vulnerabilities in our selected android apps. These three tools detect all the vulnerabilities that we are searching for in android applications. A comparison of supported tests by these tools is presented in Table I. A short overview of these three tools is presented below:

A. AndroBugs

AndroBugs [40] is a well-known test tool for checking vulnerabilities of android app. The Key features of this framework are as follows.

- Checking the existence of flaw in coding
- Dangerous shell commands detection (e.g. s̈u)
- Check security protection of an app
- Testing how the app connects with the cloud

AndroBugs has found some security vulnerabilities issues in some of the popular app.

B. SandDroid

SandDroid is another automatic Android application analysis system. It works fully online. The apk file needs to be uploaded at their website [41], where a number of tests are run in their sandbox environment to perform static, dynamic and comprehensive analysis.

The Static Analysis of SandDroid comprises of more than 8 types of analysis. Basic information extraction is used for extracting the basic application information like file size, package name, file hash, SDK version etc. Certification analysis parses certification to check if its from Android Open Source Project (AOSP) [42]. Category analysis classifies apks depending on permissions. Permission analysis extracts declared permissions and detects if they are actually used in the app or not. Component analysis lists all components of the app including Broadcast Receivers. It also checks whether the component is exported. Advertisement module analysis extracts all advertising modules and Sensitive API analysis lists all sensitive API's and the caller code path.

SandDroid has a rich set of Dynamic analysis tools. It records all network data during an app's running period, recovers data from http flow, IP distribution analysis, based on extraction of URLs and parsed IP data. File path and data is recorded by File operation monitor. Sandroid also records all sent SMS and phone calls by the app recording the SMS block behavior. Sandroid also contains Crypto operation monitor and data leakage monitor. Comprehensive analysis contains scores based on level of risk of application according to the static and dynamic analysis result. SandDroid generates a risky Behaviors Summary, listing all risky behaviors of apps.

C. QARK

Quick Android Review Kit, also known as QARK [43], is a static code analysis tool, designed to recognize potential security vulnerabilities and points of concern for Android applications. It looks for these vulnerabilities, either in source code or packaged APKs. It is an open source tool by LinkedIn [44], released at DefCon 23 and BlackHat USA 2015. The tool is also capable of creating "Proof-of-Concept" deployable APKs and/or ADB commands and exploiting many of the vulnerabilities it finds. While decompiling APKs, it automates use of several decompilers, producing a superior result to usual single decompiler results.

The set of vulnerabilities that QARK endeavors to find includes negligently exported components, intents vulnerable to interception or eavesdropping, inadequate validation of x.509 certificate, usage of sticky intents, creation and usage of world-readable or world-writeable files, activities prone to leak data. It also finds pending intents with unsafe creation, broadcast intents that are sent insecurely, private keys embedded in the source code and evaluates the used cryptography modules, potentially exploitable Webview configurations. It checks whether apps enable backup, or are debuggable, or support outdated API versions with known vulnerabilities.

V. RESULT AND DISCUSSION

We have tested twenty selected android applications from EATL app store with the selected tools. Some of the selected apps behave responsively according to the user's input, while some show informative data from either a remote server or local database. One of the selected apps, Bikalpa Pusti, is an informative app, giving a statistical data on the food calories. Another app, Learn HTML, is a tutorial app which helps to learn HTML code. Bangla Alphabet teaches children bangla alphabet with images. Bank Info shows the bank location and information. Personal Budget and Money Calculator calculates credit and debit money. Baishakh Cam customizes pictures

with baishakhi themes. Rickshaw Race Dhaka is an interesting game where one needs to race on Dhaka streets with a rickshaw. EatlStore, is an appstore where all the selected android apps can be found, is in our selected list. Exploring Bangladesh shows some interesting and historical places of Bangladesh, while Shadhinota26 shows historical information of the independence of Bangladesh.

Apart from these, we have chosen 9 popular local apps. Notepad, SMS scheduler, Bangla Dictionary are utility apps directed towards everyday usage of the users. 360 degree is a simple yet interesting game due to its brightly coloured graphics and challenges. Audacity is a marketing app, directed towards entrepreneurs and business owners. Chorki displays live news updates from major local newspapers and online news portals. FlexiPlan app is designed to help users of specific telephone operator to plan their internet, talktime and SMS usage. The app Harriken is for helping its user to discover local restaurants and providing information and reviews about them. HiFi Public is a free application that will update the user on the latest in local and international technology news, gadget reviews, start-ups, software and hardware.

A. Test Result

We have tested the selected apps with three different tools and gathered report on those. After analyzing those reports, we have tabulated the vulnerabilities for each apps. Table II, III, IV show the combined test outcomes.

According to the result of our tests in table II and III, all the chosen apps were found to have one or more vulnerabilities. From table II, Exploring Bangladesh and Virtual Radio top the list of risk with 6 vulnerabilities each. Baishakh Cam, Child Vaccine, Shadhinata26 and Rickshaw Race also tested positive to 5 vulnerabilities. On the other hand, BDJobs, Bank Info and Art Cinema Club Centers app were vulnerable to only one issue each. If we organize this data according to the vulnerabilities, it is observed that Webview vulnerability is the most common one prevalent over 13 applications, namely Learn Html, Bangla Alphabet, Virtual Radio, Baishakh Cam, Child Vaccine, Edutube, Falgun App, Find Your Friend A Valentine, Money Calculator, Shadhinota26, Rickshaw Race Dhaka, EatlStore and Exploring Bangladesh. Other dominant vulnerabilities are Database Vulnerabilities and Storage access vulnerabilities, found in 11 and 12 apps respectively.

From table III, we see these apps have lower rate of vulnerabilities than those of table II. Hifi-Public, Chorki, Audacity, Harriken and 360 degree each show 3 vulnerabilities. The other apps in this list show one or two vulnerabilities. Just like table II, we see that Webview vulnerability is a leading problem, while being closely followed by storage access vulnerability.

To strengthen our argument, we have randomly selected a few of the most popular apps on Google Playstore, namely Gmail,Cam Scanner, Youtube, NeoReader, Clean Master and Sound Cloud. After running tests on these apps with our test tools, Gmail,Youtube and NeoReader showed none of the vulnerabilities discussed in our paper. Of the other three apps,

TABLE II
SELECTED APPS AND TEST OUTCOME (EATL)

App Name	Combined Test Outcome
Bikalpa Pusti	Database Vulnerability, Vulnerability in Debug mode
Learn Html	Vulnerability in Storage Access, Webview vulnerability, Advertisement Vulnerability
Bangla Alphabet	Storage Accessing, Webview vulnerability, Vulnerability in Debug mode
Art Cinema Club Centers	Database Vulnerability
Bank Info	Database Vulnerability
Personal Budget	Vulnerability in Debug mode Database Vulnerability
Find For Me	Database Vulnerability, Advertisement Vulnerability
Bdjobs	Database Vulnerability
Virtual Radio	Vulnerability in Storage Access, Database Vulnerability, Webview vulnerability, Advertisement Vulnerability, Vulnerability in Intent, Vulnerability in Api Call
Baishakh Cam	Vulnerability in Storage Access, Database Vulnerability, Webview vulnerability, Advertisement Vulnerability, Vulnerability in Intent
Child Vaccine	Vulnerability in Storage Access, Database Vulnerability, Webview vulnerability, Advertisement Vulnerability, Vulnerability in Intent
Edutube	Vulnerability in Storage Access, Webview vulnerability, Vulnerability in Api Call
Falgun app	Vulnerability in Storage Access, Webview vulnerability, Advertisement Vulnerability, Vulnerability in Intent
Find Your Friend A Valentine	Vulnerability in Storage Access, Webview vulnerability, Advertisement Vulnerability, Vulnerability in Intent
Money Calculator	Vulnerability in Storage Access, Webview vulnerability, Advertisement Vulnerability
Shadhinota26	Vulnerability in Storage Access, Webview vulnerability, Advertisement Vulnerability, Vulnerability in Intent, Vulnerability in Api Call
Rickshaw Race Dhaka	Vulnerability in Storage Access, Database Vulnerability, Webview vulnerability, Advertisement Vulnerability, Vulnerability in Intent
EatlStore	Vulnerability in Storage Access, Webview vulnerability, Advertisement Vulnerability, Vulnerability in Intent
Exploring Bangladesh	Vulnerability in Storage Access, Database Vulnerability, Webview vulnerability, Advertisement Vulnerability, Vulnerability in Intent, Vulnerability in Api Call

TABLE III
SELECTED APPS AND TEST OUTCOME (AUDACITY)

App Name	Combined Test Outcome
360 degree	Vulnerability in Storage Access, Webview vulnerability, Vulnerability in Intent
Audacity	Vulnerability in Storage Access, Webview vulnerability, Vulnerability in Intent
Chorki	Vulnerability in Storage Access, Webview vulnerability, Vulnerability in Intent
Bangla Dictionary	Vulnerability in Storage Access, Webview vulnerability
Flexiplan	Webview vulnerability
Harriken	Vulnerability in Storage Access, Webview vulnerability, Vulnerability in Intent
Hifi Public	Vulnerability in Storage Access, Webview vulnerability, Vulnerability in Intent
Notepad	Vulnerability in Storage Access, Webview vulnerability
SMS Scheduler	Vulnerability in Storage Access, Webview vulnerability

TABLE IV
SELECTED APPS FROM GOOGLE PLAYSTORE AND TEST OUTCOME

App Name	Combined Test Outcome
Gmail	No vulnerability
Cam Scanner	Database Vulnerability
Youtube	No vulnerability
NeoReader	No vulnerability
Clean Master	Vulnerability in Intent, Database Vulnerability
Sound Cloud	Vulnerability in Intent, Database Vulnerability, Vulnerability in Debug Mode

all of them showed Database vulnerability. Clean Master and Sound Cloud showed vulnerability of Intents. The detailed analysis is given in Table IV of our paper.

To have a quick overview of the occurrence ratio of the vulnerabilities, we have created a pie-chart in 3. From the Fig. 3 we can observe that Webview Vulnerability, Storage Access Vulnerability, Advertisement Vulnerability, Database Vulnerability are common in those apps. Among them the rate of Webview Vulnerability is very high. After analyzing the data we can observe that around 68.4% of the apps have Webview vulnerability which is very harmful to the app users. And the least vulnerability that is found in the apps is debug mode vulnerability. Its occurrence rate is around 15.8%. Surprisingly SMS and phone call vulnerability are not found in the selected apps. Vulnerability in api call is also very low. Its occurrence rate is 21%.

B. Countermeasure

Vulnerabilities need to be prevented as it is a great hindrance to the app security. It can be ensured by developing apps with an extra care. Webview should be used more securely. It becomes less secure when viewing third-party apps in Webview. When ”setAllowFileAccess” attribute of a Webview is set to

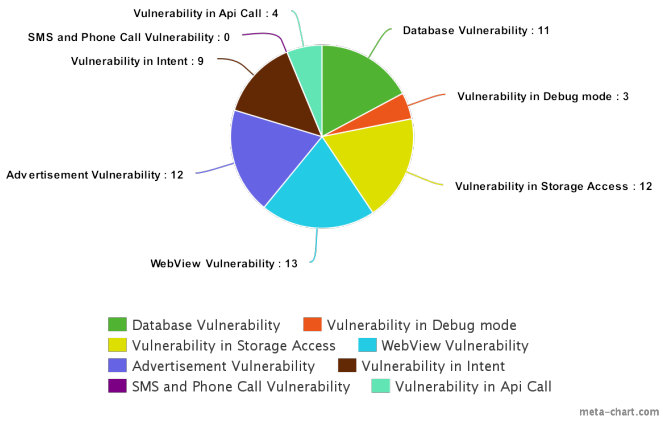

Fig. 3. Occurrence of Vulnerabilities in Selected Apps

true, attackers get the chance of accessing local storage by injecting malicious script in it. By setting setJavaScriptEnabled attribute to false, each context needs to be escaped properly by using an XSS filter component.

All the important files, database backup files, user authentication information need to store in internal storage rather than to external storage. One can view the data that is stored in the external storage without even getting inside the app.

Android debug mode needs to be turned off when releasing the app to the store. If it remains turned on then attacker may get the chance to communicate with app through adb and view the log file. To prevent database information leak one way is to use SQL Cipher [45]. Malicious advertisement can cause a user to leak his information to the hacker. If a system can be implemented that an app need to go through several vulnerabilities testing process before publishing an app in the store then a big number of vulnerabilities can be prevented.

VI. CONCLUSION

From the day when the Android was introduced, researchers have been searching for loopholes in its inherent security mechanism and they have also proposed security solution to augment the security mechanism of Android apps. However, in several cases, the vulnerabilities arise due to the lack of awareness of the developers, rather than the system flaws. In this paper, we have discussed the causes and effects of existing vulnerabilities and the tools that detect those vulnerabilities. We tested some apps from a local app store and some popular local apps from Google Play Store with three different tools that are available in the Internet. The report that we have obtained from the test are presented in a tabulated form. From the test result, we can observe that the rate of Webview vulnerability is alarmingly high. Around 13 out of the 19 apps have this vulnerability. Advertisement and Storage accessing vulnerability are also very common in these days apps. It is found in 12 apps out of the 19 apps. Hence necessary countermeasures need to be taken in no time. We have also discussed some countermeasures wherever possible, to defend

against the mentioned vulnerabilities. In this age when most of the private and sensitive data are stored digitally and are targets of cyber-criminals, it is imperative to ensure app and data safety to the full extent.

REFERENCES

[1] *Number of smartphone users worldwide from 2014 to 2019*. Available at http://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/.

[2] *Number of available applications in the Google Play Store from December 2009 to February 2016*. Available at http://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/.

[3] *23 Disturbing Statistics about Mobile Security*. Available at http://blogs.air-watch.com/2015/10/23-disturbing-statistics-mobile-security/#.V69ArxIprvs.

[4] *Malicious Android apps slip into Google Play, top third party charts*. Available at http://www.theregister.co.uk/2016/05/17/viking_horde_android_app_malware/.

[5] Victor Chebyshev and Roman Unuchek. Mobile malware evolution: 2013. *Kaspersky Lab ZAOs SecureList*, 24, 2014.

[6] Zhejun Fang, Qixu Liu, Yuqing Zhang, Kai Wang, and Zhiqiang Wang. Ivdroid: static detection for input validation vulnerability in android inter-component communication. In *Information Security Practice and Experience*, pages 378–392. Springer, 2015.

[7] K Jamsheed and K Praveen. A low overhead prevention of android webview abuse attacks. In *International Symposium on Security in Computing and Communication*, pages 530–537. Springer, 2015.

[8] DONG Guowei, WANG Meilin, SHAO Shuai, and ZHU Longhua. Android application security vulnerability analysis framework based on feature matching. *Journal of Tsinghua University (Science and Technology)*, 65(5):461–467, 2016.

[9] Da-Woon Leem, Hyun-Ju Jung, Moon-Sung Hwang, Jung-Ah Shim, and Hyun-Jung Kwon. A single-process design for developing automation tools for inspecting the vulnerabilities of android applications. 2015.

[10] *Application Security Vulnerability: Code Flaws, Insecure Code Understanding Application Vulnerabilities*. Available at http://www.veracode.com/security/application-vulnerability.

[11] Li Li, Alexandre Bartel, Tegawendé F Bissyandé, Jacques Klein, Yves Le Traon, Steven Arzt, Siegfried Rasthofer, Eric Bodden, Damien Octeau, and Patrick McDaniel. Iccta: Detecting inter-component privacy leaks in android apps. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1*, pages 280–291. IEEE Press, 2015.

[12] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *ACM SIGPLAN Notices*, 49(6):259–269, 2014.

[13] Li Li, Alexandre Bartel, Jacques Klein, and Yves Le Traon. Automatically exploiting potential component leaks in android applications. In *2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 388–397. IEEE, 2014.

[14] Xing Jin, Xuchao Hu, Kailiang Ying, Wenliang Du, Heng Yin, and Gautam Nagesh Peri. Code injection attacks on html5-based mobile apps: Characterization, detection and mitigation. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 66–77. ACM, 2014.

[15] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, and Konrad Rieck. Drebin: Effective and explainable detection of android malware in your pocket. In *NDSS*, 2014.

[16] Patrick Mutchler, Adam Doupé, John Mitchell, Chris Kruegel, and Giovanni Vigna. A large-scale study of mobile web app security. In *Proceedings of the Mobile Security Technologies Workshop (MoST)*, 2015.

[17] Siegfried Rasthofer, Steven Arzt, and Eric Bodden. A machine-learning approach for classifying and categorizing android sources and sinks. In *NDSS*, 2014.

[18] Asaf Shabtai, Uri Kanonov, Yuval Elovici, Chanan Glezer, and Yael Weiss. andromaly: a behavioral malware detection framework for android devices. *Journal of Intelligent Information Systems*, 38(1):161–190, 2012.

[19] Parvez Faruki, Ammar Bharmal, Vijay Laxmi, Vijay Ganmoor, Manoj Singh Gaur, Mauro Conti, and Muttukrishnan Rajarajan. Android security: a survey of issues, malware penetration, and defenses. *IEEE Communications Surveys & Tutorials*, 17(2):998–1022, 2015.

[20] Timothy Vidas, Daniel Votipka, and Nicolas Christin. All your droid are belong to us: A survey of current android attacks. In *WOOT*, pages 81–90, 2011.

[21] Hongliang Liang, Dongyang Wu, Jiuyun Xu, and Hengtai Ma. Survey on privacy protection of android devices. In *Cyber Security and Cloud Computing (CSCloud), 2015 IEEE 2nd International Conference on*, pages 241–246. IEEE, 2015.

[22] Long Lu, Zhichun Li, Zhenyu Wu, Wenke Lee, and Guofei Jiang. Chex: statically vetting android apps for component hijacking vulnerabilities. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 229–240. ACM, 2012.

[23] Zhemin Yang, Min Yang, Yuan Zhang, Guofei Gu, Peng Ning, and X Sean Wang. Appintent: Analyzing sensitive data transmission in android for privacy leakage detection. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 1043–1054. ACM, 2013.

[24] Bradley Schmerl, Jeffrey Gennari, Javier Cámara, and David Garlan. Raindroid–a system for run-time mitigation of android intent vulnerabilities. 2016.

[25] Shweta Bhandari, Rishabh Gupta, Vijay Laxmi, Manoj Singh Gaur, Akka Zemmari, and Maxim Anikeev. Draco: Droid analyst combo an android malware analysis framework. In *Proceedings of the 8th International Conference on Security of Information and Networks*, pages 283–289. ACM, 2015.

[26] Zhenlong Yuan, Yongqiang Lu, Zhaoguo Wang, and Yibo Xue. Droid-sec: Deep learning in android malware detection. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 371–372. ACM, 2014.

[27] Gerardo Canfora, Eric Medvet, Francesco Mercaldo, and Corrado Aaron Visaggio. Detecting android malware using sequences of system calls. In *Proceedings of the 3rd International Workshop on Software Development Lifecycle for Mobile*, pages 13–20. ACM, 2015.

[28] Shina Sheen and Anitha Ramalingam. Malware detection in android files based on multiple levels of learning and diverse data sources. In *Proceedings of the Third International Symposium on Women in Computing and Informatics*, pages 553–559. ACM, 2015.

[29] Kevin Allix, Tegawendé F Bissyandé, Quentin Jérome, Jacques Klein, Yves Le Traon, et al. Large-scale machine learning-based malware detection: confronting the 10-fold cross validation scheme with reality. In *Proceedings of the 4th ACM Conference on Data and Application Security and Privacy*, pages 163–166. ACM, 2014.

[30] Hao Zhang, Danfeng Daphne Yao, and Naren Ramakrishnan. Detection of stealthy malware activities with traffic causality and scalable triggering relation discovery. In *Proceedings of the 9th ACM symposium on Information, computer and communications security*, pages 39–50. ACM, 2014.

[31] Hugo Gascon, Fabian Yamaguchi, Daniel Arp, and Konrad Rieck. Structural detection of android malware using embedded call graphs. In *Proceedings of the 2013 ACM workshop on Artificial intelligence and security*, pages 45–54. ACM, 2013.

[32] AB Bhavani. Cross-site scripting attacks on android webview. *arXiv preprint arXiv:1304.7451*, 2013.

[33] Tongbo Luo, Hao Hao, Wenliang Du, Yifei Wang, and Heng Yin. Attacks on webview in the android system. In *Proceedings of the 27th Annual Computer Security Applications Conference*, pages 343–352. ACM, 2011.

[34] *Android WebView vulnerability allows hacker to install malicious apps*. Available at http://thehackernews.com/2013/09/android-webview-vulnerability-allows.html.

[35] *Google puts 60 percent of Android users at risk with WebView security changes*. Available at http://www.v3.co.uk/v3-uk/news/2389839/google-puts-60-percent-of-android-users-at-risk-with-webview-security-changes.

[36] *Android Hacking & Security - Part 8: Insecure Local Storage*. Available at http://resources.infosecinstitute.com/android-hacking-security-part-10-insecure-local-storage/.

[37] *Android Hacking and Security, Part 3: Exploiting Broadcast Receivers*. Available at http://resources.infosecinstitute.com/android-hacking-security-part-3-exploiting-broadcast-receivers/.

[38] *Most Android devices running outdated versions*. Available at http://www.computerweekly.com/news/4500271242/Most-Android-devices-running-outdated-versions.
[39] Bradley Reaves, Logan Blue, Dave Tian, Patrick Traynor, and Kevin RB Butler. Detecting sms spam in the age of legitimate bulk messaging. In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, pages 165–170. ACM, 2016.
[40] Yu-Cheng Lin. *AndroBugs Framework*. Available at https://www.github.com/AndroBugs/AndroBugs_Framework,version1.0.0.
[41] *SandDroid Testing Tool*. Available at http://sanddroid.xjtu.edu.cn/.
[42] *Android Open Source Project*. Available at https://source.android.com/.
[43] *Qark Testing Tool*. Available at https://github.com/linkedin/qark.
[44] *Social Network Site*. Available at https://www.linkedin.com/.
[45] Rachmawan Ardiansa. *Developing secure android application with encrypted database file using sqlcipher*. PhD thesis, UTeM, 2014.