Journal of Information Systems Engineering & Management, 1:3 (2016), 187-190
ISSN: 2468-4376 eISSN: 2468-4376

Vulnerabilities Classification for Safe Development on Android

Ricardo Luis D. M. Ferreira*, Anderson F. P. dos Santos, Ricardo Choren
Instituto Militar de Engenharia, BRAZIL
*Correspondence to: ricardo.ldm.ferreira@gmail.com; anderson@ime.eb.br; choren@ime.eb.br

ABSTRACT

The global sales market is currently led by devices with the Android operating system. In 2015, more than 1 billion smartphones were sold, of which 81.5% were operated by the Android platform. In 2017, it is estimated that 267.78 billion applications will be downloaded from Google Play. According to Qian, 90% of applications are vulnerable, despite the recommendations of rules and standards for the safe software development. This study presents a classification of vulnerabilities, indicating the vulnerability, the safety aspect defined by the Brazilian Association of Technical Standards (*Associação Brasileira de Normas Técnicas* - ABNT) norm NBR ISO/IEC 27002 which will be violated, which lines of code generate the vulnerability and what should be done to avoid it, and the threat agent used by each of them. This classification allows the identification of possible points of vulnerability, allowing the developer to correct the identified gaps.

Keywords

Android,
Mobile Applications,
Security,
Vulnerability,
Vulnerability Classification,
Bad-Practices

DOI: 10.20897/lectito.201634

INTRODUCTION

Android is currently the most used mobile platform in the world. Devices with Android operating system dominate the smartphone market, representing 81.5% of the market (Llamas et al, 2015)

TABLE 1. VOLUME OF UNITS SOLD IN 2015.

Operating System	2014	
	Volume *	Marketshare
Android	1059.3	81,5%
IOS	192.7	14,8%
Windows Phone	34.9	2,7%
BlackBerry	5.8	0,4%
Others	7.7	0,6%
Total	1,300.4	100,0%

* Units in Millions

Despite all efforts to ensure the security of information on these devices, 97% of the free apps and 80% of other applications for Android, selected for evaluation by the company ARXAN (2014), have some vulnerability. According to Qian, 90% of applications are vulnerable. These vulnerabilities may lead to the impairment of important safety aspects: Confidentiality, Integrity and Availability.

However, there are norms and standards that recommend ways to ensure that the software development process is safely performed. The Brazilian Association of Technical Standards *(*Associação Brasileira de Normas Técnicas - *ABNT)* norms ISO/IEC 27002 and ISO/IEC 15408 stand out in relation to security in the process of software development. In addition, other initiatives contribute to improve safety in applications such as BSIMM-V (Building

Security in Maturity Model), SEI CERT Coding Standards (Software Engineering Institute - Carnegie Mellon University) and Best Practices of Secure Coding OWASP Quick Reference Guide (OWASP Secure Coding Practices - Quick Reference Guide).

The existence of these norms and standards do not guarantee the security quality of applications published in the official stores. Therefore, this study presents a vulnerability classification which considers: the vulnerabilities identified by the project OWASP Mobile Security Project; classifies them according to the ABNT NBR ISO/IEC 27002 safety aspects that may be violated and the types of code to Android development that generate this vulnerability, indicating whether the risk should warn or stop the mobile application development process.

ANDROID

Android is an operating system developed to be used in mobile platforms. It is the most currently used, for being an open platform (Scarsella et al, 2015). This platform has an architecture based in layers, in which a superior layer uses services provided by the lower layers.

The *Kernel Linux* layer is the core of the operating system of the platform. Responsible for the low level services, allow access to the system components. The hardware and network drivers (sound, cameras, USB, wifi, GPS, among others), file and processing systems. In the *Libraries* layers are the platform`s native library, such as APIs (*Application Programming Interface*) for the presentation layer, the database management, among others. *Android Runtime* allows the applications to be executed in mobile devices. In the *Application Frameworks* are the APIs which allows the applications to use the resources of the platform, such as telephony, location and notifications. Finally, in the Applications layer are the applications that run on the platform. May be native applications such as the address book, or developed by third parties.

In addition to these layers, the knowledge of the components that are part of an application is needed in order to develop a software for the Android platform. The Activities implement a type of interface where the user information is displayed, manage and coordinate the flow of events in the screens. Services is the component responsible for tasks running in the background, usually tasks that require a longer time to run. The platform allows information to be shared between applications. An example would be an application that creates a new contact in the phonebook. The feature of the Content Management platform (Content Providers) will be used for this. The Broadcast Receivers is another component. This component is responsible for identifying the occurrence of a particular event native or triggered by an application.

Finally, the AndroidManifest.xml manifest, which is unique to each application, has the settings and information of which components that will be used by the application. Android Core, which is the Android platform itself, is responsible for the interaction between the components and applications.

INFORMATION SECURITY

Information security is the protection of information and assets of an organization against various threats, in order to preserve the confidentiality, integrity and availability (Associação, 2013).

Confidentiality is the property, which will limit the access to information only for those users with permission to view or change it.

Integrity is the property that only authorized processes will change the information.

Finally, the Availability is the property that ensures that an information is always available to authorized users.

OWASP

Open Web Application Security Project (OWASP) is a open community dedicated to enable the organizations to develop, acquire and maintain reliable applications. Its mission is to make individuals and businesses aware of the risks involved in the development of secure software. Among the companies that collaborate with OWASP are Adobe, HP, Salesforce, EY, Akamai, BlackHat, Contrast Security, Symantec, UPS, Aspect Security, Reackspace and TrendMicro.

OWASP Mobile Security Project is a security project for mobile applications with a focus on security checks. It is intended for developers and security teams and aims to provide the resources to build and maintain secure mobile applications, classify the risks and provide the necessary controls to minimize the impacts or exploitation of vulnerabilities. Keeps the list of the Top Ten Mobile Risks OWASP vulnerabilities, which features those that were analyzed and classified by this organization in 2014 (OWASP, 2014).

The ten vulnerabilities listed in the last version published in 2014 by the OWASP Top Ten Mobile Risks are:

M1. *Weak Server Side Controls:* To build APIs and web services and make available for other applications is common. In some cases, these APIs or services may contain serious safety problems when an unsafe code is used.

When a software uses this service in its architecture, the applications that are on the server side should contain security measures to ensure and prevent unauthorized users from accessing sensitive information of the applications.

According to OWASP, the most frequent problems that allow the exploitation of this issue are the authentication failures, unsafe settings, injections (SQL, XXS and Commands), the access control vulnerability and session management flaws.

M2. *Insecure Data Storage:* Related to the loss or exposure of user data that are stored on the user's device. May occur in the form of exposure of confidential data, logs vulnerabilities or transaction history. The storage data on the device is not recommended in order to avoid this type of vulnerability. Any data on the device should only be stored if it is really necessary.

M3. *Insufficient Transport Layer Protection:* Occurs when the network traffic between the mobile application and the application server does not provide protection. If this vulnerability is successfully exploited, allows data to be intercepted by the attacker. All information sent from the device must be encrypted and use a secure channel such as HTTPS whenever possible in order to minimize this risk.

M4. *Unintended Data Leakage:* Occurs when the developer, without prior knowledge, saves data or sensitive information in a location of the mobile device that is easily accessed by other applications. The vulnerabilities of the Operating System (OS), hardware and frameworks are included.

M5. *Poor Authorization and Authentication:* The exploitation of this vulnerability occurs when the attacker knows how the application authentication process works and, from this moment, makes a false authentication request to the system. Once authenticated, it performs the operations available without being noticed.

OWASP recommends, in order to avoid this type of attack, that: requests for authentication are performed on the server, whenever possible; weak passwords or the device id should not be allowed by the system for authentication; and that the user credentials are not stored on the device, otherwise, they must be encrypted.

M6. *Broken Cryptography:* Occurs when the encryption used by the developer is very weak and this allows the attacker to easily decipher the encrypted information travelling over the application.

The recommended measures to prevent such attacks are: never save on the device, important credentials or data; disable key logging; and not use insecure encryption algorithms.

M7. *Client Side Injection:* Results in the execution of malicious code on the device when the application receives a malicious or malformed data entry. SQL injection, for instance, is based on a simple text to be interpreted and loaded by the application without performing any validation.

The best way to prevent this attack is to ensure that all application data entries are validated before running an operation with these data.

Injection Attacks in mobile applications are in the form of SQL injection, Inclusion in Local File, Java Script Injection (XSS), Interface Injection and Attack on binary code.

M8. *Security Decisions Via Untrusted Inputs:* The Android platform has a mechanism which allows, the communication between applications through the exchange of messages, Inter-Process Communication - IPC. This vulnerability occurs when an application sends a message with untrusted parameters to another application and it does not perform a validation of the information received.

In addition, Android's permission system allows the developer to decide which permissions the application should have. During the installation, the user decides whether the permissions informed are in accordance with the objective of the application. Despite the apparent security that it offers, an inexperienced developer can request more permissions than the application actually needs. Thus, another malicious application can communicate with *this application* and use these permissions to steal data.

The creation of a list of trusted applications that can exchange messages with the application; the prevention of sensitive data exchange via IPC and the validation of data entries; and the occurrence of interaction with the user when a navigation from one application to another occurs; are recommended to prevent the exploitation of this vulnerability.

M9. *Improper Session Handling:* Session tokens are used to maintain the status of applications. The server sends a session cookie to the application in a successful authentication. This cookie should be added to future transactions between the application and the server.

When the session token is obtained by a threat agent during a transaction between the mobile application and the server, this agent can impersonate a legitimate user using the stolen token in future transactions and thus gain an advantage with the information obtained.

The vulnerability that allows improper session handling appears as follows:

- When the application timeout is not well defined;
- With the incorrect invalidation of the session on the server. The developer usually invalidates the session only in the application;
- With the change in the user profile when the cookie does not reflect this change; and

- Finally, when the tokens creation is not safely done.

M10. *Lack of Binary Protections:* An attack, which exploits this vulnerability, modifies the source code of a program through reverse engineering. Its goal is to create a dangerous application that run malicious code, however, disguised as a safe application, which does not raise suspicions when opened by the user.

The use of checksum control techniques and the debugger detection control, among others, avoid this attack. The use of techniques to prevent the reverse engineering of the code and the detection in runtime of change in the code by the application, are other prevention methods.

CLASSIFICATION OF VULNERABILITIES

The purpose of this study is to classify the vulnerabilities of the OWASP Top Ten Mobile Risks list, in order to help developers to create and maintain safer codes, showing what might happen when an unsafe code is developed.

The identification of which risks mentioned by the OWASP are likely to be preventively avoided, during the development process and that were directly related to the mobile device, was necessary to achieve this goal. Vulnerabilities that were associated to the server side, for instance, were discarded. Then, the safety aspects defined by the norm ABNT NBR ISO/IEC 27002 which would be infringed if the risk was exploited and the attack vector used, were determined. After this, the types of code for Android that generate these vulnerabilities were pointed out. Finally, this classification indicates to the developer if the risk is a case of warning or interruption.

The vulnerabilities M1, M3, M6 and M10 are not part of the scope as they are exploited after the process of application development, when it is ready and published in an official store. M1 does not apply, because the prevention should occur on the server side; M3 refers to security problems at the network layer; M6 arises from encryption breaking; and M10 is related to the reverse engineering of the program.

TABLE 2. CLASSIFICATION OF VULNERABILITIES

Risk	Type of Codes		Type	Threat Agents	Violated Aspect
M2-Insecure Data Storage	SharedPreferences	getSharedPreferences / getPreferences	Alert	Physical access to the mobile device / malware / malicious apps	Confidentiality / Integrity
	Internal Storage	openFileOutput			
	External Storage	getExternalStorageDirectory + FileOutputStream			
M4- Unintended Data Leakage	HTTP request cache on internal storage	getCacheDir + HttpResponseCache.install	Alert	Physical access to the mobile device / malware / malicious apps	Confidentiality
	HTTP request cache on shared/external storage	getExternalCacheDir + HttpResponseCache.install			
M5- Poor Authorization and Authentication	Input type password	inputType="textPassword"	Alert	Malware / botnets	Confidentiality / Integrity / Availability
	SharedPreferences	getSharedPreferences / getPreferences			
	Internal Storage	openFileOutput			
	External Storage	getExternalStorageDirectory + FileOutputStream			
M7- Client Side Injection	Designing queries for SQLite	Insert / insertOrThrow / insertWithOnConflict / delete / update / updateWithOnConflict / query / queryWithFactory / rawQuery / rawQueryWithFactory / replace / replaceOrThrow	Error	Users / malicious apps	Confidentiality / Integrity / Availability
	Javascript permission	setJavaScriptEnabled / AddJavascriptInterface / setPluginState	Alert		
M8- Security Decisions Via Untrusted Inputs	Dynamic permission via call IPC mechanism	checkCallingOrSelfPermission	Alert	Users / malicious apps / malware	Confidentiality / Integrity
M9- Improper Session Handling	Invalidate cookies with CookieManager class	removeAllCookies / removeSessionCookies	Alert	Physical access to the mobile device / malware	Confidentiality
	Invalidate cookies with CookieStore class	remove / removeAll			

A. *Insecure Data Storage*

As an attack aimed at exploring the Insecure Data Storage vulnerability is successful, the information security aspects of Confidentiality and Integrity are violated. This is due the obtention of confidential data by an unauthorized user, who can even modify them.

Depending on the information obtained the attacker can steal user identities, commit fraud or harm the reputation of the user, such as, when a successful attack can illicitly obtain access credentials and the attacker assumes the user's identity.

This usually occurs because the developer assumes that anyone with bad intentions will get access to the device where the application is installed. The fact that the performance of improper actions in the device in case of loss, theft of installation of malware, is ignored.

The developer should avoid storing sensitive information on the device in order to prevent the exploitation of the system breach for this vulnerability. Otherwise, the resources of getSharedPreferences and getPreferences classes should be reminded to be avoided. If their use is needed, the private access mode (MODE_PRIVATE) must be used, in order to ensure that only the application that created the preferences file or other applications that have the same user id will have access to this preference. The MODE_WORLD_READABLE and MODE_WORLD_WRITEABLE modes, which have become obsolete in the API 17, should not be used. The first allows any application to access the file for reading and the second allows write access.

Likewise, the important data storage should not use the internal memory of the device. If the internal memory is used by the openFileOutput method, the same warning must be given to the developer, indicating the same

considerations regarding the access mode and adding that the use of the setStorageEncryption native API will add some security, given that the data will be encrypted.

Finally, the external memory, such as an SD card, is a precarious storage type and its use should alert the developer that writing in this kind of media allows any entity (a person or another application), to have access to the file, and that if the card is removed, the data is lost. In addition, suggest the use of cryptography API, such as javax.crypto. This type of vulnerable code is found through the file path identification using the getExternalStorageDirectory method, followed by the opening of the file for writing with FileOutputStream code.

B. Unintended Data Leakage

There are situations in which the developer of an application decides to store HTTP requests data in cache in order to gain performance. However, similar to what happens with Insecure Data Storage, an agent can obtain access to the device, either through a malware, a maliciously modified application, or even physical access to the device, obtaining these cached information, causing unintended leak of data.

If this occurs, the attacker now in possession of legitimate user information, can commit fraud or damage to the user's reputation, since their privacy has been violated, thus breaking the concept of Confidentiality.

Therefore, the developer must be alerted when the HttpResponseCache.install code is used to store the cache of requests. This code must be preceded by the directory identification where the cache is stored. If the storage occurs in the internal memory, HttpResponseCache.install should be preceded by getCacheDir. If it occurs in the external memory, getExternalCacheDir will be used. In this case, the free access of the external memory is important to be considered, since it can be rewritten by any entity or be removed and not be accessible.

C. Poor Authorization and Authentication

A flawed process of authentication and authorization allows a malicious agent to steal information, cause damage to the user's reputation and commit fraud on his behalf. Once this attack is successful, the Confidentiality, Integrity and the Availability of information will be lost.

The attacker installs a malware on the device or uses botnets they created in order to exploit this vulnerability. Having control over the device and knowing the authentication process of the application, a false authentication request is performed and then, the services of the system are requested. Among the many recommendations that exist to minimize the impact of this risk, the most are related to the authentication process on the server side. Thus, the initiatives that will help the developer to improve its implementation considering the scope of work are below.

When the developer creates an Activity that has a field of type inputType = "textPassword" an alert to use a password validation method that does not allow weak password, such as only 4 digits, should be given.

Furthermore, the same considerations made for the vulnerability Insecure Data Storage can be applied in case the developer locally stores authentication information, which would not be a good programming practice.

D. Client Side Injection

Consider that a legitimate user of the application or another malicious application sends invalid entries to be processed by the application. Assume that this application does not validate the data entries received before using them for processing or storing. Under these conditions, an attacker who obtains or changes information he should not have access, through a code injection attack, the Confidentiality, Integrity and Availability of this information will be lost.

The developer must be alerted or warned, depending on the situation, that the code being created will produce a breach for this risk to be explored, in order to prevent the theft of confidential information, fraud, or violation of user data privacy.

The SQL injection occurs when a threat agent enters malicious data in fields of an Activity and then the system performs a query in the SQLite database without being subjected to a validation process. The application development must be stopped and the code should be corrected identifying the insert methods, insertOrThrow, insertWithOnConflict, delete, update, updateWithOnConflict, query, queryWithFactory, rawQuery, rawQueryWithFactory, replace, replaceOrThrow, the SQLiteDatabase class, whether they are using parameterized queries or are not validating input data in order to prevent the SQL injection from occurring. This validation can be done through a blacklist of characters that can not be used in the input data.

Another attack, known as XSS (Cross-site scripting), allows scripts to be injected and executed in the application. When the developer uses the setJavaScriptEnabled method in a WebView, the Javascript code is authorized to be run. Similarly, the use of AddJavascriptInterface method inserts a JavaScript code in the WebView that will run on the next page load. When the developer uses these methods, he should be warned that, from this moment, the application can suffer XSS attacks. This is valid for the setPluginState method that although discontinued from the API 18, allows the execution of a plug-in WebView.

E. Security Decisions Via Untrusted Inputs

By exploiting this vulnerability, the attacker, who can be a user, a malware or a malicious application, violate the Confidentiality and Integrity of the information contained in the device.

The developer, when using the checkCallingOrSelfPermission method should be warned because this method allows both the application and the process of origin of an IPC call to verify whether the system has a given permission, for example, view the phone contacts. In this case, a malicious application that does not have explicit permission for this could have access to contacts through this new application, setting the leak of permission.

F. Improper Session Handling

Similarly to the Poor Authorization and Authentication, a failed authentication process allows the information theft, the damage to the user's reputation and fraud on the user's behalf. Thus, the Confidentiality of the information is destroyed.

In this scenario, an attacker with physical access to the device or from a malware, can obtain, through session cookies stored on the device, the user's access credentials. Thus, this threat agent could impersonate the user to perform actions on his behalf without arousing suspicion.

Among the recommendations suggested to prevent this vulnerability to exist, many of them reference actions that must be taken in the application server side. However, the developer of an application should be alert when saving session cookies on the device. When the developer uses the removeAllCookies and removeSessionCookies methods from the CookieManager class, or remove and removeAll of CookieStore class, he should be warned that these cookies are invalidated on the application server side. This is necessary to prevent that these cookies remain valid on the server side, opening a breach so that third parties use this information.

CONCLUSIONS

This study presents a classification of vulnerability for the Android platform. A risk classification was presented in which, the types of codes that make an application vulnerable, as well as which aspects of security will be violated and what should be done to prevent the occurrence of these vulnerabilities were identified from the OWASP Top Ten Mobile Risks list.

This study differs from Qian et al. (2015) and Wu et al. (2014) as they analyze the vulnerabilities from the dex bytecode of the application, when it is already published in an app store. In these cases, the approach will be corrective, while in this study, a preventive action approach is presented. Regarding Souza (2015), this study differs in the platform used, since Souza used a web environment with Java programming. Besides, Android allows the use of resources available on the mobile device; also, the interface mapping of an object, created in XML (Extensible Markup Language), which manipulates attributes and behaviors through classes and methods specific to this platform.

REFERENCES

ARXAN Technologies. State of Mobile App Security: Apps Under Attack. (2014). Available at: https://www.arxan.com/wp-content/uploads/assets1/pdf/State_of_Mobile_App_Security_2014_final.pdf. (Acessed 21 July 2015).

ASSOCIAÇÃO Brasileira de Normas Técnicas. (2013). ABNT ISO/IEC 27002:2013 - Information Technology - Security Techniques - Code of Practice for Information Security Management. (*ABNT ISO/IEC 27002 - Tecnologia da Informação - Técnicas de Segurança - Código de Prática para a Gestão da Segurança da Informação*).

LLAMAS, Ramon; Reith, Ryan; Nagamine, Kathy. Smartphone OS Market Share. (2015). Available at: http://www.idc.com/prodserv/smartphone-os-market-share.jsp. (Acessed 11 July 2015).

OWASP, Mobile Security Project - Top Ten Mobile Risks. (2014). Available at: https://www.owasp.org/index.php/Projects/OWASP_Mobile_Security_Project_-_Top_Ten_Mobile_Risks. (Acessed 09 November 2014).

QIAN, Chenxiong; LUO, Xiapu; LE, Yu; GU, Guofei. (2015). 'VulHunter: Toward Discovering Vulnerabilities in Android Applications', IEEE Computer Society, Hong Kong, The Hong Kong Polytechnic University, p. 363-376.

SCARSELLA, Anthony; REITH, Ryan; SHIRER, Michael. (2015). Worldwide Smartphone Market Will See the First Single-Digit Growth Year on Record, According to IDC. International Data Corporation. Available at: http://www.idc.com/getdoc.jsp?containerId=prUS40664915. (Acessed 19 Februery 2016).

SOUZA, Luciano Sampaio Martins de. (2015). 'Early Vulnerability Detection for Supporting Secure Programming', Rio de Janeiro, Pontifícia Universidade Católica do Rio de Janeiro.

WU, Daoyuan; LUO, Xiapu; CHANG, Rocky KC. (2014). 'A Sink-driven Approach to Detecting Exposed Component Vulnerabilities in Android Apps', Hong Kong, The Hong Kong Polytechnic University, arXiv preprint arXiv:1405.6282.