Actas de las segundas Jornadas Nacionales de Investigación en Ciberseguridad

Granada, 15, 16 y 17 de junio de 2016

Universidad de Granada

Eds.: Pedro García Teodoro, Rafael A. Rodríguez Gómez y Fco. Javier López Muñoz.

ISBN: 978-84-608-8070-7

Security Assessment Methodology for Mobile Applications

Álvaro Botas, Juan F. García, Javier Alonso
RIASC, University of León, Spain
León, Spain
abotm,jfgars, jalol@unileon.es

Jesús Balsa, Francisco J. Lera,
Christian García, Vicente Matellán
University of León, Spain
León, Spain
jbalc,fjrodl,cgarcl,vmato@unileon.es

Raúl Riesco
INCIBE
León, Spain
raul.riesco@incibe.es

***Abstract*—Any type of software, from desktop to mobile applications, is prone to contain defects that can lead to vulnerabilities. These vulnerabilities, when exploited, may put in risk the integrity, confidentiality and availability of the software. Security auditing methodologies help to reduce at some level of confidence these risks. With the explosion of mobile applications for daily activities like checking email, news, social networks, or even managing bank accounts, guaranteeing an acceptable level of application security becomes critical for the usage and trust of mobile services. In this paper, we review and classify OWASP 2014 Top Ten mobile risks in analysis blocks. Based on the blocks classification, we propose a methodology to security audit mobile software applications. We demonstrate the effectiveness of the proposed methodology by auditing the same mobile application in Google's Android and Apple's iOS platforms surfacing multiple vulnerabilities.**

***Index terms*—security, auditing, methodology, OWASP, Android, iOS**

Tipo de contribución: *Investigación en desarrollo*

I. Introduction

Any software system is prone to contain a non-negligible fraction of remaining defects when released. These defects caused by different factors like the growing pervasiveness and complexity of the software or the increasing market pressure to deploy new services and features as soon as possible limiting the time dedicated for testing. Aside of the potential consequences of these defects in terms of reliability and availability largely discussed in the literature [1], [2], these defects may lead to software vulnerabilities. We understand a software vulnerability as software flaws that might be accessible to an attacker who can wittingly exploit them [3]. Examples of common software vulnerabilities are buffer overflows, cross-site scripting (XSS) and SQL injection, among others [4].

Developing *vulnerability-free* software might be impossible to achieve due to time and economical restrictions. However, there exist different methods to surface software vulnerabilities, such as data flow analysis, taint analysis or fuzzy testing, for naming a few [5]. The more vulnerability detection techniques are used, the higher the probability that the software has less remaining vulnerabilities. However, the application of these techniques are usually expensive and burdensome.

Due to the exponential growth of mobile applications ecosystem, there are thousands of applications available to conduct any type of daily activity, from checking email, news, or social networks to managing bank accounts. Therefore, users rely on these applications sensitive data. Hence, any vulnerability in mobile applications can put in risk the user's privacy. Besides, mobile applications ecosystem promotes the first one-of-a-kind application released. So, mobile applications are usually released to the market as soon as possible, castigating critical features like security and reliability [1], [2].

In order to cope with vulnerabilities and security threats in software products, security auditing methodologies have been proposed to certify certain security level of confidence. To the best of our knowledge, there are neither recognised consensus nor standards about how to perform a security audit in mobile applications. However, there are general best practices like the IEEE standard for software reviews and audits [6]. However, these general best practices are not considering the idiosyncrasy of mobile applications.

The most closed approach to develop some best practices or guidelines to develop and maintain secure mobile applications is managed by Open Web Application Security Project (OWASP). OWASP offers developers and security teams resources to build and maintain secure mobile applications [7], but no formal guideline is provided. However, it is important to recognise the effort conducted by OWASP in order to list the most representative risks during mobile application development.

In this paper, we introduce an auditing methodology for mobile applications covering the risks pointed out by OWASP. We firstly review, analyse and classify the OWASP 2014 Top Ten Mobile Risks [7] under different analysis blocks. Then, we propose a methodology that uses these analysis blocks to audit the security of mobile applications. The methodology is evaluated in practice by auditing the same functional application developed for two different mobile platforms: Google's Android and Apple's iOS. We focus on these platforms since they lead the global smartphone market [8]. Our methodology revealed several application design flaws related to implementation and handling of data, as well as information leakage. We found that sensitive application user data may be easily reachable by an attacker or third-party applications, even remotely.

The outline of this paper is as follows. Section II visits the related work. Section III summarises the OWASP Top Ten Mobile Risks, and the analysis blocks identified. The methodology proposed is introduced in Section IV. A case of use is presented and discussed in Section V. Section VI

concludes the paper.

II. Related Work

As noted earlier, to the best of our knowledge, there is not a methodology for security auditing on mobile applications. Recent published works are mainly focused on the application usability [9] or rapid development [10], [11], rather than security aspects of the application itself. Nevertheless, it is worth to mention [12], where a methodology to empirical analysis of permission-based security models and enhancements to Android is introduced.

Although, several tools are available to analyse concrete security aspects – mainly related with the confidentiality – in mobile platforms, they have specific usages. However, they can be used together to conduct a security audit in mobile applications. For instance, ProtectMyPrivacy [13] detects access to private data and substitutes it by anonymised data under user requests in Apple's iOS. Similarly, other tools, such as PiOS [14], PSiOS [15], DiOS [16], perform static or dynamic analysis into applications to detect sensitive data leakage.

Likewise, TaintDroid [17] monitors privacy data using an information flow tracking system. SCanDroid [18] allows to check whether data flow through an application is consistent with its security specifications. Other tools, such as DroidScope [19], Andrubis [20], and Google's Bouncer [21], focus on analyzing mobile applications to detect malicious behaviors (i.e., malware).

III. Analysis Blocks to Identify Mobile Risks

In this section, we summarize the OWASP Top Ten Mobile Risks for 2014 [7] to identify where the common defects are found and how they can be surfaced during an application security auditing. The OWASP Top Ten Mobile Risks list the most common risks according to different factors such as threatening agents, attack vectors, weaknesses, technical impact, and business impact. OWASP classifies common mobile security risks and provide assurance controls to reduce its impact or likelihood of exploitation. The OWASP risks are briefly introduced:

(M1) *Weak Server Side Controls* considers the risks from third-party components like backend servers required by most mobile applications. This risk includes insecure server configuration, authentication errors, session management weakness, access control vulnerabilities, just to mention few.

(M2) *Insecure Data Storage* considers the potential risks caused by vulnerabilities on the data storage in the mobile device. These risks can cause information leakage.

(M3) *Insufficient Transport Layer Protection* identifies vulnerabilities like non-encrypted transport layer communications, the usage of weak crytographic algorithms or the acceptance of unauthorized certifications.

(M4) *Unintended Data Leakage* risk considers the unknown potential vulnerabilities in the data management by the application or operating system.

(M5) *Poor Authorization and Authentication* considers risks associated with unacceptable authentication assumptions like that only authenticated users can send requests to the server without further validation of the user or weak authentication protocols.

(M6) *Broken Cryptography* includes the incorrect usage of the encryption/decryption process or the usage of weak in nature cryptographic algorithms.

(M7) *Client Side Injection* gathers the risks of not validating user input data, avoiding code injection.

(M8) *Security Decisions via Untrusted Inputs* brings together the risk associated with accepting any type of input source. This is specially relevant in the case of Inter Process Communication (IPC) mechanisms.

(M9) *Improper Session Handling* collects the defects that can lead in vulnerabilities in handling user session.

(M10) *Lack of Binary Protection* identified as risk using untrusted source repositories to host the mobile application code. Any source repository out of physical control by the organization has to be considered insecure.

Considering the target of the attack vector, these risks can be grouped into those which objectives are the server and communications, and those targeting the application itself.

Risks affecting server and communications include M1, M3, M5, M6 and M9. All these risks have in common that they are considering risks in the communication between the mobile application and the backend servers or lack of poor authorization/authentication in the server side when it is receiving client requests. For instance, self-signed or expired certificates, vulnerable SSL implementations, or encrypted communications weak and even unencrypted.

Those targeting the application are M2, M4, M7, M8 and M10. These risks refer to vulnerabilities like data application files containing sensitive data (such as password or username) stored in plain text and accessible by third-party applications, unknown OS vulnerabilities that may unintendedly compromise an application, and the deployment of executable code without additional protections, such as obfuscation or packing.

We have identified five analysis blocks which can detect a risk area according to application logic. Namely, the blocks are: *Environment Analysis*, *Connections*, *Sensitive Data*, *Application Own Data*, and *Application Structure*. Each block clearly delimits the aspects to be analysed in a mobile application to detect likely vulnerabilities. The risks of OWASP can belong to several blocks of analysis as OWASP classifies risks in light of the threats beyond that analysis purposes. For instance, the M5 risk accounts for authentication not only on the application side (i.e., a user must be logged in to access to concrete backend services), which belongs to *Application Structure*; but also on the server side (i.e., a backend service must not be anonymously reached), which is covered by *Environment Analysis*. Similarly, M5 is also covered by *Application Own Data* since an application may ask for a weak login password to execute.

Environment Analysis: This analysis block covers the OWASP Top Ten Mobile Risks M1, M5, M7, and M10. The environment analysis block encompasses the study of the mobile device (i.e., if the device is using a custom or modified firmware, platform versions, etc.), the application backend server, and the information related to the application (i.e., the market from which it was downloaded, who developed it, whether it can be decrypted/reversed, etc.).

Application backend servers are used by mobile applications for different purposes: Either to provide extra func-

tionalities, to be customized/updated, or to properly execute. Although remote servers might be considered to be out of scope when auditing an application, note that application data can be compromised whether these servers are compromised somehow. Thus, this analysis block verifies whether confidentiality and integrity properties are hold.

Connections: This block relates to OWASP Top Ten Mobile Risks M1, M3, M4, and M7. It encompasses the analysis of all types of communications available in the mobile device, and thus likely to be used by an application. They can be from device to device, device to the Internet or device to computer. Communications can be performed using different technologies, such as mobile data, Wi-Fi, IRDA, Bluetooth, or NFC.

Four main questions are addressed by this analysis block:

- *Where is the application connecting to?*: The destination of any connection performed by the application has to be located and logged. For instance, the IP address or URL when connecting through the Internet, or the target device name and MAC when connecting though a Bluetooth, IRDA, or NFC connection.
- *What is the protocol being used on the connection?*: The protocol involved in any connection has to be known to detect whether is vulnerable or otherwise secure. For instance, an Internet connection using HTTP instead of HTTPS protocol enables unencrypted instead of encrypted communication.
- *What data are being transmitted in the connection?*: Whether user data are being set, we need to know what specific files are being sent, what commands are being used, etc.
- *Does the application perform certificate pinning?*: Otherwise, even though the application could perform a presumably secure connection, it could be compromised using self-signed certificates. An analysis of applications potentially vulnerable to Man-in-the-middle attacks on SSL/TLS connections is introduced in [22].

Sensitive Data: This block only relates to OWASP Top Ten Mobile Risks M4. Sensitive data includes data managed by the operating system, such as data related to device hardware Identification (UDID, MAC, IMEI, ...), its status (Wi-Fi and Bluetooth on/off, 2G/3G/4G connection, etc.), or other data related to native OS applications (contacts, calendar events, messaging, etc.).

It is common that mobile applications request access to (or even send) sensitive user data. For instance, *Path* [23], *Hipster* [24] or *Twitter* [25]) applications.

A relevant question arises: *What is the motivation behind these data accesses?*. Normally it is just for legitimated or commercial purposes, although malicious applications get this kind of data for spurious reasons. A study performed over Android applications [20] shows that, in general, applications requesting a high number of permissions are likely malicious software (i.e., malware).

Application Own Data: This analysis block fits to OWASP Top Ten Mobile Risks M2, M4, M5, M6, and M9. Application own data refers to data generated and managed by the application itself. For instance, format files such as XML, PList, SQLite are commonly used to store the application own data. These data are generally saved in files in the device, and may include credentials, images, documents, logs, or any other kind of data. So, any person having physical access to the device, a third-party application or a malware application installed on the device, could access this data. Thus, any data generated/handled by an application must be securely saved to prevent unauthorised access.

	Environment Analysis	Connections	Sensitive Data	Application Own Data	Application Structure
M1	√	√			
M2				√	√
M3		√			√
M4		√	√	√	√
M5	√			√	√
M6				√	√
M7	√	√			√
M8					√
M9				√	√
M10	√				√

Table I
RELATIONSHIPS BETWEEN OWASP'S RISKS AND THE ANALYSIS BLOCKS.

Application Structure: This analysis block covers all risks on the OWASP Top Ten except M1. This point covers everything related to the design and implementation of the application. In fact, this analysis block uses the binary code disassembled or source code of the application retrieved in the *Environment Analysis* block. Here, an analysis of the source/binary code is required.

Any vulnerability/operability analysis performed on the application is also covered by this block. This is indeed one of the most important blocks, since it provides information related to the application. For instance, the application must work properly and according to what it is expected to do. This analysis block detects when an application shows an unexpected behaviour, drains the battery or crashes during execution due to an exception. Apart from a dynamic application analysis, a static analysis is also carried out. Hard-coded strings must be studied to check whether they contain relevant information, such as developer credentials, URLs, or databases passwords.

Table I summarises the relationships between the analysis blocks and the OWASP risks.

IV. A SECURITY ASSESSMENT METHODOLOGY

In this section, we present a methodology to conduct security audit of mobile applications covering the analysis blocks presented in Section III, and thus, the OWASP Top Ten Mobile Risks. This methodology allows finding vulnerabilities, to detect malicious behaviours, and even to analyse application performance. The methodology also aims at covering a broad number of aspects when auditing a mobile application, such as detecting coding flaws or insecure design.

The methodology consists of three main phases subdivided into several tasks: *Pre-runtime*, which encompasses the analysis of the application before executing it; *Runtime*, which describes the analysis performed to an application under execution; and *Post-runtime*, which encompasses the analysis tasks after an application execution. Fig. 1 depicts a flowchart of the proposed methodology.

A. Pre-Runtime Phase

This phase is performed before executing the application. It can be divided into two tasks: *Preliminary Analysis*, and

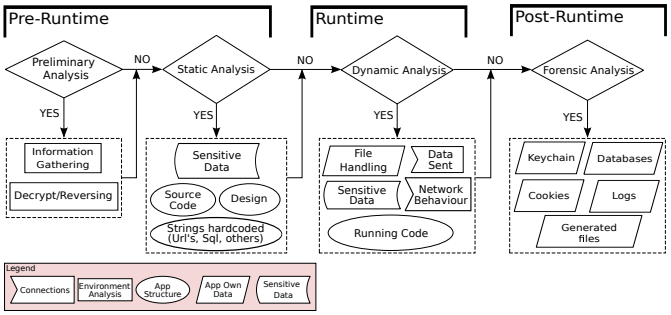

Figure 1. Flowchart of the proposed methodology.

Static Analysis. The former is related to Environment Analysis block, while the latter applies to Application Structure.

The Preliminary Analysis task describes the process of information gathering, decrypting, and reversing the application. Information gathering is the process of obtaining information about the application. Information can be passive, when it refers to the information collected non-intrusively from the device or remote servers (e.g., developer's, application description, or server information), or active, when the information is collected intrusively. For instance, scanning tools are used in the remote servers to obtain the IP, OS, public services and potential vulnerabilities. Decrypting and reversing are the processes to fully obtain the binary code in a (somehow) human-readable form. For instance, Java classes and related content are obtained when analysing a Google's Android application. Similarly, assembly code is obtained when analysing an Apple's iOS application.

The Static Analysis is decomposed into multiple tasks, such as analysing all strings (readable text) retrieved from the source code, binary code, or configuration files, studying the source/binary code to find vulnerabilities (e.g., code coverage analysis, authorization logic schemes, or quality test), analysing the design of the application to check its abstraction, modularity, or extensible considerations, or to analyse the sensitive data accessed by the application. Note that the source code or the disassembled binary code is needed.

B. Runtime Phase

This phase is performed during the execution of the application. It can be divided into several tasks: *File Handling*, *Data Sent*, *Access to Sensitive Data*, *Network Behaviour*, and *Running Code*. In this phase, Connections, Application Own Data, and Sensitive Data analysis blocks are involved.

File Handling covers the process of writing, reading and handling (i.e., creation, deletion or movement) files within the application using either third-party applications or native functions of the OS. For instance, whether an Android application stores a file on the external memory, this file could be read by any other application. However, when the file is stored within the application sandbox, it is only accessible by the own application. Unlike Google's Android, Apple's iOS always stores files in the application sandbox, which restricts unauthorised file access.

Data Sent refers to the network packets being transmitted by the application. Third-party applications such as Wireshark, Burp, or `tcpdump` can be used in this step, independently from the OS in which the application is running on. To check whether the connection is ciphered, is using certificate pinning and credentials are also sent ciphered are usually performed in this phase. An auditor also must check whether the data being sent is sensitive or private, and verify this sending is done with user consent.

Sensitive Data focuses on verifying to what data the application is accessing at runtime, since these data could differ from the results obtained in the Pre-Runtime phase. For instance, an application can create new code during execution, and make use of a buffer overflow to eventually access to this new code and access to some sensitive data [26]. An auditor can use specific tools (e.g., TaintDroid [17] or DiOS [16]) to monitor system calls related to data handling.

Network Behaviour is related to the analysis of Internet connections, i.e., the final destinations of any connection established by the application. These connections might be legitimate when it is related to the service provided by the application, or illegitimate when referring to exfiltrating user behaviour or other sensitive user data.

Running Code covers the entire process of dynamically analysing an application. To check whether an application is performing the role it should do is a part of these analysis phase. Messages, activities, services, events, etc. must be monitored by an auditor, and keep tracking of methods executed, parameters, and return values.

C. Post-Runtime Phase

The last phase of the methodology deals with forensics data, as it is performed after the application has been executed. Namely, we propose to analyse keychain, cookies, property lists, databases, logs and files downloaded during the execution of the application. This phase includes only the *Data analysis block*.

Keychain is the internal credential storage managed by the OS, usually very secure unless the device has a non-native OS firmware (i.e., it is rooted or jailbreaked). An application with high privileges can read system memory, and thus read the keychain. Therefore, it is recommended to store credentials in ciphertext instead of plaintext.

Cookies are used by the OS to connect to services without further login intentions. For instance, in Apple's iOS, the cookies are stored in the backup files, therefore an attacker can access to different services using a stolen authenticated session cookie.

Property List and Configuration Files correspond to generated files which store any kind of sensitive data, private information or even credentials. These files are sometimes stored in plain text in insecure locations and in the backup files. Thus, they can be usually accessed without privileges, and thus, stolen.

Databases, normally SQLite, refer to files generated during execution and used to store internal application data. These data can be very diverse, such as private data, sensitive data, or credentials. An auditor must review all these files to verify what they contain, how it is stored and handled.

Logs describes the files that contain information of the execution, usually used for debugging purposes. These files are useful to analyse the reasons of application failures, or the status of its execution.

Finally, Files Downloaded relate to files that are downloaded via any connection (e.g., JavaScript files, binaries, configuration files, etc.). These files must be studied to verify its type and what for they are being used.

V. Case Study: eCommerce Application

In order to evaluate the effectiveness of the methodology proposed, we audit the same application on Google's Android and Apple's iOS platforms. We have chosen an application (for anonimity called eCommerce app) from a Spanish online outlet, which is freely available at Google Play and Apple Store. eCommerce app is a multi-platform application used by more than 15 million of world-wide users, containing several functionalities where the methodology applies: Authorisation, authentication, user session handling, sensitive data handling, user private data storage, backend data handling, purchases, etc.

For the sake of space, we only reveal the most significant vulnerabilities found during the auditing process. A full report can be found at [27].

A. On Pre-Runtime Phase

The first step is to gather information about eCommerce app. Passive information matches for both platforms: The application author is a known developer named *Anonimised*; servers and user databases used by eCommerce app coincide those used by the eCommerce website utilizes; this also happens to other services such as databases of users and others. Active information shows a CVE-2004-0230 [28] vulnerability on the server es.ecommerce.com: A Denial-of-Service attack on this server would totally leave the application out of service.

A static analysis on eCommerce app reveals the application permissions. In the case of Google's Android version, the declared permissions are related to the use of the Internet and the Google Cloud Messaging [29], to vibrate, to keep the device on, to access to user's Google account registered on the device, and to read phone contacts and phone state. In our opinion, the last permissions could be considered non relevant for this kind of application. Permissions declared in Apple's iOS version clearly differ: Apart from Internet access, it has access to the MAC address, location information, pasteboard and advertising IDs. In our opinion, again some of these information is not necessarily needed.

Besides, some of these permissions could be used to harvest user data. For instance, the application could periodically monitor the pasteboard for changes and read any copy-pasted sensitive data out. We conclude that in both platforms the application asks for some permissions without a clear purpose.

Analysing the strings in both platform applications, we have found references to login passwords stored in MD5, to data stored in SQLite3 files, as well as the login email in plain text. This issue is relevant for the next phase.

B. On Runtime Phase

In this phase we deeply analyse the network behaviour and data sent by eCommerce app in both platforms. Both applications are communicating with the same servers where vulnerabilities aforementioned in login and shopping requests.

```
POST /auth/login HTTP/1.1
cookie: force_platform=web; dtCookie=2620DC1711659F8DF749DC4A8B742FCB|_default|1;
TS669e14=867f89f5a3c728490653d5680e3e55415b9e27161cb99b35544248a883971f5f152444d421
9396cde0a53e77; _ga=GA1.2.430789342.1413630153;
        JESSID_es=f0uq3dlsunvvl9dkats65rqfc1;
TS5eb823=1b2d1ffd72e7c471df188d2c37e7cd5d2baf62ff5e138a2f54b7882b;
TS252493=49a86fccfb3179bdebcc7e59ab2bd9dd5b9e27161cb99b3554b788025ef7bf877ee7511d
Content-Length: 126
Content-Type: application/x-www-form-urlencoded
Host: es.        .com
Connection: Keep-Alive

login_type=md5&source=mobile&member_login_email=floro_2012%40hotmail.es&member_logi
n_password=020494b6                              44ca4c4
```

(a) HTTP POST login request in Google's Android platform

```
POST /mycart/checkout HTTP/1.1
Host: es.        .com
Proxy-Connection: keep-alive
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Cookie: TS252493=5adfdaabd78bd73f843d0ac2a363bc7fc280c1c9b2e73c2a54b7904d; dtCookie=|_default|0;
        JESSID_es=ns97vetbrm6ujodf3mmbmafb25;
TS669e14=6fa0c1575802d00375ffc65f9d380aebc280c1c9b2e73c2a54b7904d83971f5f973c7e0f219396cd2bcfe94d
Accept-Language: es-es
Accept: */*
Content-Length: 264
Connection: keep-alive
User-Agent:          /2.8 CFNetwork/672.1.13 Darwin/14.0.0

card_holder_name=Ramon&card_number=5784725484544096&card_expiration_month=01&card_expiration_year
=2016&cvv2=437&payment_method=mastercard&member_advertising=on&amex_zipcode=&commercial_name=&bus
siness_id=&accept_conditions=on&billing_dir=&billing_cp=&billing_city=
```

(b) HTTPS POST payment request in Apple's iOS platform

Figure 2. Vulnerable (a) login and (b) payment requests of eCommerce app.

Login requests are performed using an unencrypted HTTP connection and sending plain-text username and password encrypted with MD5 out. Note that MD5 has been repeatedly demonstrated to be vulnerable [30], [31]. Besides, a large number MD5 rainbow tables can be found on the Internet, so an attacker has a high chance of getting back the password in plain text. Every time a user opens the application and logs in, even when the login credentials are remembered, the login request is sent. Figure 2(a) depicts the HTTP POST login request generated by eCommerce app in Google's Android platform.

eCommerce app uses mobile retargeting, i.e., it collects information on the mobile user to later send to the user personalised advertising. This feature is offered by several third-party companies. In this case, it uses a TapCommerce service and sent relevant data out, such as session ID, timestamp, and OS.

Purchase requests are done using encrypted HTTPS connections but without certificate pinning. This means that the application is vulnerable to self-signed certificates, which allow to decrypt data being sent. An Man-in-the-Middle (MitM) attack using a transparent proxy such as Mallory could be used towards this end.

Similarly, payment requests use an encrypted HTTPS connections but without certificate pinning. The data being sent out are related with credit card used for payment: Card number, card issuer, card holder, card expiration, and CVV code. Note that an attacker can retrieve these sensitive data using a similar scenario to the purchase request attack. Figure 2(b) depicts the HTTPS POST payment request generated in Apple's iOS platform.

File handling analysis shows that files are written to the application sandbox, which prevent access by third-party applications.

During the execution of the application, the most-used permissions in Apple's iOS platform are the access to the MAC address, pasteboard and the iPhone's Unique Identifier (UDID).

C. On Post-Runtime Phase

In the case of eCommerce app, the main issue is to verify how the data are stored since it is a benign application.

Credentials data are stored outside the OS keychain in both platforms. However, files written into the application sandbox are plain text and contain these credentials. In the case of the Google's Android version, the user email is stored in plain text while the user's password is stored in MD5. In the case of Apple's iOS version of eCommerce app stores the user email and login password in plain text and this file is saved when backuping. Thus, this information can be read by any attacker with access to the backup files. This problem not only endangers the confidentiality of user credentials but also other services used by the user, since people often use the same credentials for different services (e.g., email, social networks, cloud storage, etc.).

Lastly, it is worth mentioning that a cookie of eCommerce website is stored on Google's Android platform containing session information.

Findings Summary

In brief, we found several vulnerabilities in each phase of the proposed methodology: During the Pre-Runtime phase, we found a vulnerability on the remote servers used by the application that can lead to unavailability; On Runtime phase, we found that user credentials are transmitted in plain text but password, which is weak ciphered using MD5. Moreover, purchase requests containing credit card user data are transmitted using a encrypted connection but likely to be spoofed by a MitM attack; On Post-Runtime we found files generated by the application are totally insecure stored, being easily accessed by third-party applications, by malware, or by physically accessing to the device or a device backup file. We have notified all vulnerabilities the spanish CERT to be eventually corrected.

VI. Conclusions

Today mobile applications process tons of private end user data due to its pervasive usage for any daily activity. Hence, it is necessary to develop methodologies to guarantee the security in mobile applications. Therefore, a security auditing methodology is needed to verify that a mobile application is doing what is expected to do, and no hidden behaviours are detected. In this paper, we consider the OWASP Top Ten Mobile Risks, and based on them, we define five main analysis blocks that cover these risks (*Environment Analysis*, *Connections*, *Application Own Data*, *Sensitive data*, and *Structure of the application*). Using these blocks, we propose a methodology to guide an auditor to perform the audit of a mobile application. By applying it, the auditor can certify that the mobile application covers the top risks identified by OWASP as critical.

This methodology allows finding vulnerabilities, and detecting suspicious behaviours in a mobile application. We validate our methodology analysing a widely used spanish online outlet application available for Google's Android and Apple's iOS platforms. In both platforms several vulnerabilities have surfaced. These vulnerabilities may compromise the confidentiality, integrity of the user data and the availability of the service provided by the application.

Agradecimientos

This work was partially supported by the Spanish National Institute of Cybersecurity (INCIBE) accordingly to the rule 19 of the Digital Confidence Plan (Digital Agency of Spain) and the University of León under the contract X43. Also, we would like to thank to Ricardo J. Rodriguez for helping us with the development and desing of the paper.

References

[1] K. Ewusi-Mensah, *Software Development Failures*. MIT Press, 2003.
[2] R. Charette, "Why software fails [software failure]," *IEEE Spectrum*, vol. 42, no. 9, pp. 42–49, September 2005.
[3] O. H. Alhazmi, S. Woo, and Y. K. Malaiya, "Security vulnerability categories in major software systems," in *Procs. IASTED 2006*, 2006, pp. 138–143. [Online]. Available: http://www.cs.colostate.edu/~malaiya/pub/CNIS-547-097.pdf
[4] M. Dowd, J. McDonald, and J. Schuh, *The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities*. Addison-Wesley, 2006.
[5] W. Jimenez, A. Mammar, and A. Cavalli, "Software Vulnerabilities Prevention and Detection Methods: A Review," in *Procs. SEC-MDA 2009*. CTIT Workshop Procs. Series, Jun 2009, pp. 6–13. [Online]. Available: http://www.utwente.nl/ewi/ecmda2009/workshops/ECMDA2009-SEC-MDA.pdf
[6] IEEE STD 1028-2008, "IEEE Standard for Software Reviews and Audits," IEEE STD 1028-2008, pp. 1–52, 2008.
[7] OWASP, "OWASP 2014 Top Ten Mobile Risks," [Online; accessed at January 28, 2014], January 2014, https://www.owasp.org/index.php/OWASP_Mobile_Security_Project.
[8] International Data Corporation, "Smartphone OS Market Share, Q3 2014," [Online; accesed at November 28, 2014], 2014, http://www.idc.com/prodserv/smartphone-os-market-share.jsp.
[9] M. G. Cimino and F. Marcelloni, "An efficient model-based methodology for developing device-independent mobile applications," *J. Syst. Architect.*, vol. 58, no. 8, pp. 286–304, 2012.
[10] Y.-J. Jeong, J.-H. Lee, and G.-S. Shin, "Development Process of Mobile Application SW Based on Agile Methodology," in *Procs. of ICACT 2008*, Feb 2008, pp. 362–366.
[11] V. Rahimian and R. Ramsin, "Designing an Agile Methodology for Mobile Software Development: A Hybrid Method Engineering Approach," in *Procs. of RCIS 2008*, June 2008, pp. 337–342.
[12] D. Barrera, H. G. Kayacik, P. C. van Oorschot, and A. Somayaji, "A Methodology for Empirical Analysis of Permission-based Security Models and Its Application to Android," in *Procs. of ACM CCS'10*. New York, NY, USA: ACM, 2010, pp. 73–84.
[13] Y. Agarwal and M. Hall, "ProtectMyPrivacy: Detecting and Mitigating Privacy Leaks on iOS Devices Using Crowdsourcing," in *Procs. of MobiSys '13*. New York, NY, USA: ACM, 2013, pp. 97–110.
[14] M. Egele, C. Kruegel, E. Kirda, and G. Vigna, "PiOS: Detecting Privacy Leaks in iOS Applications," in *Procs. of NDSS 2011*. The Internet Society, 2011. [Online]. Available: http://www.isoc.org/isoc/conferences/ndss/11/pdf/9_2.pdf
[15] T. Werthmann, R. Hund, L. Davi, A.-R. Sadeghi, and T. Holz, "PSiOS: Bring Your Own Privacy & Security to iOS Devices," in *Procs. ASIA CCS'13*. New York, NY, USA: ACM, 2013, pp. 13–24.
[16] A. Kurtz, A. Weinlein, C. Settgast, and F. C. Freiling, "DiOS: Dynamic Privacy Analysis of iOS Applications," Friedrich-Alexander-Universität Erlangen-Nürnberg, Dept. of Computer Science, Technical Reports CS-2014-03, June 2014.
[17] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "TaintDroid: An Information Flow Tracking System for Real-time Privacy Monitoring on Smartphones," *Commun. ACM*, vol. 57, no. 3, pp. 99–106, Mar. 2014. [Online]. Available: http://doi.acm.org/10.1145/2494522
[18] A. P. Fuchs, A. Chaudhuri, and J. S. Foster, "SCanDroid: Automated Security Certification of Android Applications," Department of Computer Science, University of Maryland, College Park, Tech. Rep. CS-TR-4991, November 2009.
[19] L. K. Yan and H. Yin, "DroidScope: Seamlessly Reconstructing the OS and Dalvik Semantic Views for Dynamic Android Malware Analysis," in *Procs. of USENIX SEC'12*. Berkeley, CA, USA: USENIX Association, 2012, pp. 29–29.
[20] M. Lindorfer, M. Neugschwandtner, L. Weichselbaum, Y. Fratantonio, V. van der Veen, and C. Platzer, "ANDRUBIS - 1,000,000 Apps Later: A View on Current Android Malware Behaviors," in *Procs. BADGERS 2014*, 2014, to appear.

[21] H. Lockheimer, "Android and Security," [Online; accesed at March 21, 2014], February 2012, http://googlemobile.blogspot.com.es/2012/02/android-and-security.html.

[22] S. Fahl, M. Harbach, T. Muders, L. Baumgärtner, B. Freisleben, and M. Smith, "Why Eve and Mallory Love Android: An Analysis of Android SSL (in)Security," in *Procs. CCS'12*. ACM, 2012, pp. 50–61.

[23] A. Thampi, "Path uploads your entire iPhone address book to its servers," [Online; accessed at October 1, 2013], 2012, http://mclov.in/2012/02/08/path-uploads-your-entire-address-book-to-their-servers.html.

[24] M. Chang, "Hipster uploads part of your iPhone address book to its servers," [Online; accessed at November 7, 2013], February 2012, http://blog.markchang.net/post/17244167951/hipster-uploads-part-of-your-iphone-address-book-to-its.

[25] D. Sarno, "Twitter stores full iPhone contact list for 18 months, after scan," [Online; accessed at December 3, 2013], February 2012, http://articles.latimes.com/2012/feb/14/business/la-fi-tn-twitter-contacts-20120214.

[26] T. Wang, K. Lu, L. Lu, S. Chung, and W. Lee, "Jekyll on iOS: When Benign Apps Become Evil," in *Procs. USENIX SEC'13*, ser. SEC'13. Berkeley, CA, USA: USENIX Association, 2013, pp. 559–572.

[27] Álvaro Botas, "Security Audit Report of eCommerce App," Research Institute of Applied Sciences in Cybersecurity, Tech. Rep., 2015, available at http://grupos.unileon.es/riasc/files/2015/01/eCommerceAuditReport.pdf.

[28] CVE Details, "Vulnerability Details : CVE-2004-0230," [Online; accessed at January 5, 2015], August 2004, http://www.cvedetails.com/cve/CVE-2004-0230/.

[29] Google, "Google Cloud Messaging for Android," [Online; accessed at January 5, 2015], https://developer.android.com/google/gcm/index.html.

[30] X. Wang and H. Yu, "How to Break MD5 and Other Hash Functions," in *Procs. EUROCRYPT 2005*, ser. LNCS, R. Cramer, Ed., vol. 3494. Springer, 2005, pp. 19–35.

[31] M. Stevens, A. Sotirov, J. Appelbaum, A. Lenstra, D. Molnar, D. Osvik, and B. de Weger, "Short Chosen-Prefix Collisions for MD5 and the Creation of a Rogue CA Certificate," in *Procs. CRYPTO 2009*, ser. LNCS, S. Halevi, Ed., vol. 5677. Springer, 2009, pp. 55–69.