# **Mobile App Security Assessment Plan**

Zaid Fadel

Grand Canyon University

SWE-550-O500

30-Nov-2022   
**Document Version 1.0**



.

This Paper prepared for SWE-550-O500

Table of Contents

[**Mobile App Security Assessment Plan** 1](#_Toc120783118)

[Software Security Assessment Data Flow Diagram Plan: 3](#_Toc120783119)

[Impact of Open Source Threats: 3](#_Toc120783120)

[**ANALYSIS BLOCKS TO IDENTIFY MOBILE RISKS in Analysis and requirements phase:** 3](#_Toc120783121)

[Threat Action Plan: 4](#_Toc120783122)

[• **Where is the application connecting to?:** 4](#_Toc120783123)

[• **What is the protocol being used on the connection?:** 4](#_Toc120783124)

[• **What data are being transmitted in the connection?:** 4](#_Toc120783125)

[• **Does the application perform certificate pinning?:** 4](#_Toc120783126)

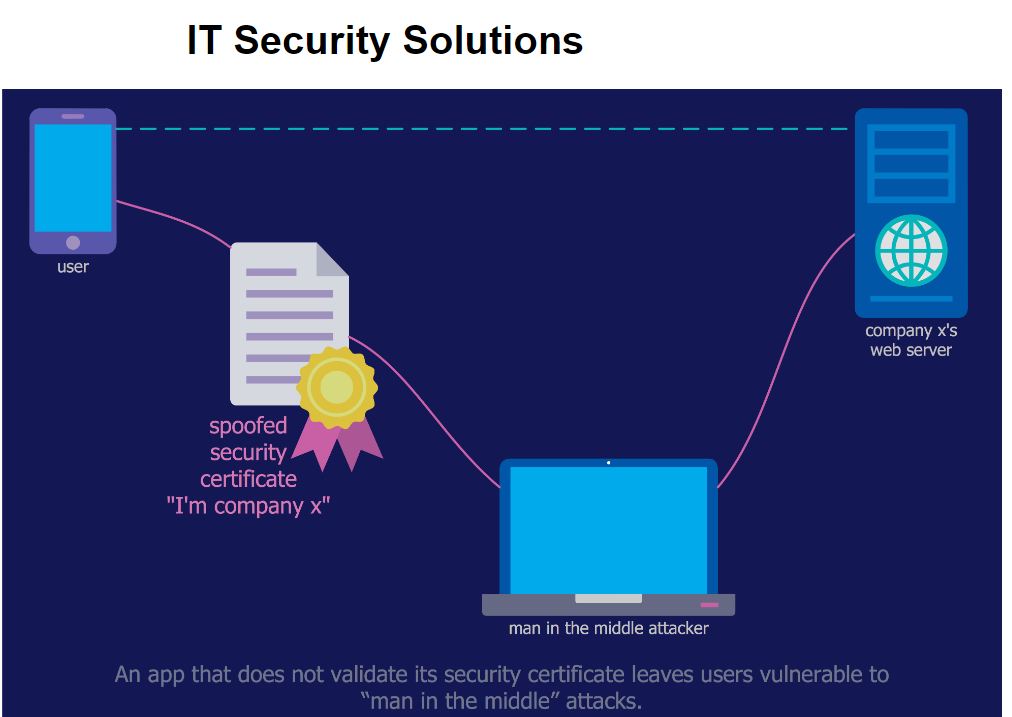
[**Pre-Runtime(Analysis) Phase of SDLC:** 4](#_Toc120783127)

[**Development Phase (Runtime Phase):** 5](#_Toc120783128)

[**Post-Runtime Phase** 5](#_Toc120783129)

[**CONCLUSIONS** 5](#_Toc120783130)

# Software Security Assessment Data Flow Diagram Plan:



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 i

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 contribuci´

on: Investigac

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 avail-

ability 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 impos-

sible to achieve due to time and economical restrictions.

Howeve

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 avail-

ability 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 impos-

sible to achieve due to time and economical restrictions.

Howeve

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, crosssite 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.

# Impact of Open Source Threats:

## **ANALYSIS BLOCKS TO IDENTIFY MOBILE RISKS in Analysis and requirements phase:**

**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.

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

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

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

**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

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

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

# Threat Action Plan:

* **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]

# **Pre-Runtime(Analysis) Phase of SDLC:**

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

**Development Phase (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.

**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

**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, 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).