AN AUTOMATED TOOL FOR DETECTION AND ENFORCEMENT OF SECURITY IN MOBILE APPLICATION DEVELOPMENT

P. A. I. U. Amarasekera

148202X

Degree of Master of Computer Science

Department of Computer Science and Engineering

University of Moratuwa
Sri Lanka

May 2018

AN AUTOMATED TOOL FOR DETECTION AND ENFORCEMENT OF SECURITY IN MOBILE APPLICATION DEVELOPMENT

P. A. I. U. Amarasekera

148202X

This dissertation submitted in partial fulfillment of the requirements for the Degree of Master of Computer Science specializing in Mobile Computing

Department of Computer Science and Engineering

University of Moratuwa

Sri Lanka

May 2018

DECLARATION

I declare that this is my own work and this MSc. project report does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement and declaration are made in the text.

Also, I hereby grant to University of Moratuwa the non-exclusive right to reproduce and distribute my thesis, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works.

---------- ----------

P.A. I. U Amarasekera Date

I certify that the declaration above by the candidate is true to the best of my knowledge and that this project report is acceptable for evaluation for the MSc. research project.

---------- ----------

Dr. Malaka Walpola Date

ABSTRACT

With the large number of mobile applications being developed and used, the mobile application security has become a key concern to the mobile application users as well as to the mobile application designers, developers and testers. Numbers of security guidelines and prevention mechanisms have been introduced through previous research work and considerable amount of mobile security frameworks, testing tools and source code analyzers have been implemented upon those research outcomes. However it was identified that these tools and instruments majorly support the testing phase of secure software development life cycle and there is a research gap open for developing a technically supportive program for the developers to build secure mobile applications.

The intention of this project is to come up with a concept where the developer is enforced to build a secure mobile application based on a predefined set of security criteria during the application development phase. These security criteria are defined based on security requirements of the mobile application project. The source code will be validated against these security criteria and if any issue is found, it will be fixed automatically during the source code compilation. This system is implemented in java platform with the help of java annotation processor and xml parser. The source code is written as s a set of reusable jar file which is published as “buildsec” library. This library is tested and evaluated in android mobile platform by injecting vulnerable codes snippets into the android mobile source code and “buildsec” library was able to find and fix those security issues in the source code. The automatic fixing of security issues during compile time will help the development team to ensure that the mobile application is security compliance in advance. This will reduce the testing effort as well as development re-work that takes to fix the security issues originated from the development phase.

ACKNOWLEDGEMENTS

I would like to start off by expressing my deepest love and affection to my parents and family for their love and immense support, which has been a constant source of strength throughout this journey.

My deepest gratitude and admiration goes to my supervisor, my mentor; Dr. Malaka Walpola, for his invaluable guidance by providing extensive knowledge, materials, advice and supervision throughout this research work. His expertise and continuous guidance enabled me to accomplish my research. Furthermore my appreciation goes to Mr. Amodth Jayawardena and Mr. Prasad Sooriyaarachchi for providing valuable resources, technical support and advice in the area of this research.

Last but not the least; I express my profound love and admiration to all my colleagues for their invaluable help on finding relevant research material, sharing knowledge and experience and for their generous encouragement.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation	Description
API	Application Program Interface
APK	Android Package Kit
APN	Access Point Name
CIA	Confidentiality, Integrity, Availability
DEX	Dalvik EXecutable
DHS	Department of Homeland Security
DNS	Domain Name System
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hyper Text Transfer Protocol with Secure Sockets Layer
IP	Internet Protocol
IPA	IPhone Application Archive
IPC	Inter Process Communication
JAR	Java ARchive
MAM	Mobile Application Management
MDM	Mobile Device Management
MFA	Multi-Factor Authentication
NIST	National Institute of Standards and Technology
OS	Operating System
OWASP	Open Web Application Security Project
PIN	Personal Identification Number
QA	Quality Assurance
QARK	Quick Android Review Kit
SAST	Static Application Security Testing
SD	Secure Digital
SDK	Software Development Kit
SDLC	Software Development Life Cycle
SFR	Security Functional Requirements
SMS	Short Message Service
SQL	Structured Query Language
SSC	Software Security Checklist
SSL	Secure Sockets Layer
SSLC	Secure Software Life Cycle
TFA	Two-Factor Authentication

TLS	Transport Layer Security
UAT	User Acceptance Testing
UDP	User Datagram Protocol
UI	User Interface
URL	Universal Resource Locator
UUID	Universal Unique Identifier
WiFi	Wireless Fidelity

CHAPTER 1
INTRODUCTION

1.1 Background

With the rapid movement of computing towards mobile platforms the security attacks and malware have also moved their targets to mobile computing platforms [1]. As concluded by Sophos [2] in 2013 devices that run Android is the prime target for security attacks, and as reported by F-secure [3], mobile malware samples indicates a growth over 50,000 starting from few hundreds, within just two years. Furthermore, the report says that the ubiquity and the vast usage of mobile devices are the main two factors which have lead the mobile application security to be a persistent issue up until now. As users store large amounts of sensitive and personal information in the mobile devices such as personal and credit card details; mobile devices have now become the most potential and easy targets for attackers who seek financial gain. According to Symantec [4] 57% of the adults who use mobile devices are not even aware that security solutions are available for security issues in mobile applications or devices. Reports also show that around half of the security attacks are triggered to track users to steal their personal information. The limited computational power and the restricted user interface of mobile phones also create a fragile environment for attackers to hide their malicious activities during security attacks.

A security environment in mobile application is composed of three main components i.e., mobile device security, operational level security and usage environment security. Device security is where securing the mobile device in access level. Many users attempt to root their mobile devices and obtain super user access rights for full control and customization without having a proper knowledge on the negative security effects that they bring to their own devices. Most of the users do not bother to protect their devices by PIN code or biometric authentication mechanism such as fingerprint in device or application level, leaving easy access to anyone who has stolen their mobile device. Operational level security focuses on malicious behavior in mobile operating systems and applications running on the mobile OS. Currently, there are lots of tools available to

identify these malicious applications by testing the application package or analyzing the flaws in the source code (APK in Android and IPA in iOS etc.,) Any vulnerability caused during this security level will lead to lots of data losses and other issues [5]. These security issues can be prevented by securing the software application development process [6, 7]. The level of security provides in smart apps stores where the applications are distributed to the end users is discussed under the environment security [5]. Even though Apple has taken a number of steps to ensure the quality and safety of the applications developed for iPhones and tabs, Google still seem to struggle with malicious application being uploaded to "Play Store". According to Google's statistics claim that 0.16 % of the apps that users attempted to install from the "Play Store" in 2015 were found to be malicious. However in cooperate world, most of the companies and organizations now leverage mobile management services such as Mobile Device Management (MDM) and Mobile Application Management (MAM) tools to provision the enrollment of mobile devices with secure settings and centrally manage both personal and business mobile application security in order to minimize the threat of security in mobile environment [14].

The core topic of this research is mobile application security and is discussed under operational level security. The highest percentage of the security threats and vulnerabilities are caused by malicious mobile applications. As shown by several recent studies, the risk which is caused by not integrating security into the software development life cycle will badly impact the company reputation. On the other hand, "Citigal" and "Stake" who have conducted studies in this area have proved that a company is guaranteed to be benefited in both cost and reputation if they put effort to integrate security into the software development life cycle [9, 10, 11].

There are different tools and mechanisms available to identify security vulnerabilities during the testing phase. Most of these tools and instruments which are used in the life cycle are intended to focus on ensuring the reliability and safety of the application.

These tools make use of well-known testing techniques such as modeling, code auditing, testing through fault/attack trees, investigating fault injection, property-based testing and boundary testing [7]. In high-level, those testing tools can be categorized into three main categories as given below.

1. Static testing tools that analyze the application binary or the source code to identify vulnerabilities in the code usually associated with dataflow and buffer handling [17].

2. Dynamic testing tools that allow testing the behavior of a running system to identify the potential issues [17]. Usually these tools enable to simulate the actual environment in which the application is deployed. The most common type of such tools are proxies that support web services while allowing to observe the communications between mobile application clients and potentially change them during testing [17].

3. Forensic testing tools that allow examining the artifacts such as source code, external files and third party plugins which are left behind once the application is compiled and run [17]. These source code scanners/analyzers check for potential issues in the source code that would cause security risks.

Having to fix application post attack is not only financially expensive but also damages the company reputation. The proven solution is to prevent revenue loss by integrating security into the development life cycle [8]. Therefore if an organization takes appropriate steps to document their security requirements, security policies, security guidelines and procedures as well to educate and train the development and application support teams, it would be a more cost effective approach than having to fix the security issues later.

The mobile software development life cycle consists of the following phases. These phases are performed repeatedly during the application development process.

- Requirements
- Design
- Implementation
- User Acceptance Testing (UAT) / Quality Assurance (QA)
- Production

Security testing is typically performed at the QA or UAT phase closer to the end of the development life cycle. Limiting security testing to one or two phases of the SDLC also limits coverage of security. Hence, development teams which follow this approach often end up not identifying security defects in the source code until the end of development life cycle or until the code is compiled and deploy as a functional application. When security defects are discovered at later stages of the development life cycle the entire process become highly inefficient and fixing these issues will also be quite expensive [16].

However, so far there is no effective tool or mechanism available for developers to support through development process to ensure that all identified vulnerabilities are addressed in the code they develop and release to the testing team. Even though that developing robust and vulnerability free software is a challenging job [6], it is purely the individual developer's responsibility to manually incorporate the security guidelines while developing the app which is not clearly reliable.

Hence this research is to come up with a concept where the developer is ensured to build a secure application based on a predefined set of security guidelines before it is shipped for testing. This will reduce the testing effort as well as development re-work that take to fix the security issues find out during the testing phase.

1.2 Problem Statement

As a strategic initiative to develop secure mobile applications, it is essential to incorporate security into every phase of the mobile Software Development Life Cycle (SDLC). However, with the large number of mobile applications being used in the constantly changing threat landscape, it is critical for an organization to utilize different means of test approaches for security analysis to ensure complete security coverage for SDLC. Usually, there are security gaps between application design and the approved corporate policies which would be exposed during security assessments. These security gaps are typically evolved during application development or integration of different modules [18]. A static application security testing on the mobile source code throughout the development stage will be an effective approach to assist developers to ensure secure application development [16].

Therefore, the main research problem trying to address through this research study is to develop an open source build tool in which one part of it can work as a static analyzer to find the vulnerabilities in the source code and another part of it can work on automatically fixing all security issue tracked by the analyzer, based on secure coding guidelines and pre-defined set of vulnerability criteria based on the security requirements of the project. Currently there are several commercial and open source tools which are capable of validating and assessing the security of a mobile application during different phases of SDLC. There are few source code analyzers which are capable of finding security vulnerabilities in the source code but not able to automatically fix them. Most of the other tools are used in testing phase when the application has already been built and released for testing. No matter how many tools are available for security testing as long as the developers inject security vulnerabilities to the code, the security of the application cannot be guaranteed and there will always be more iterations running between the testing and development phases which will ultimately cost to the project in effort and time. Hence, the proposed tool will be

integrated into the development phase or the environment of the SDLC to find and automatically fix the security vulnerabilities in the source code during development.

1.3 Objectives

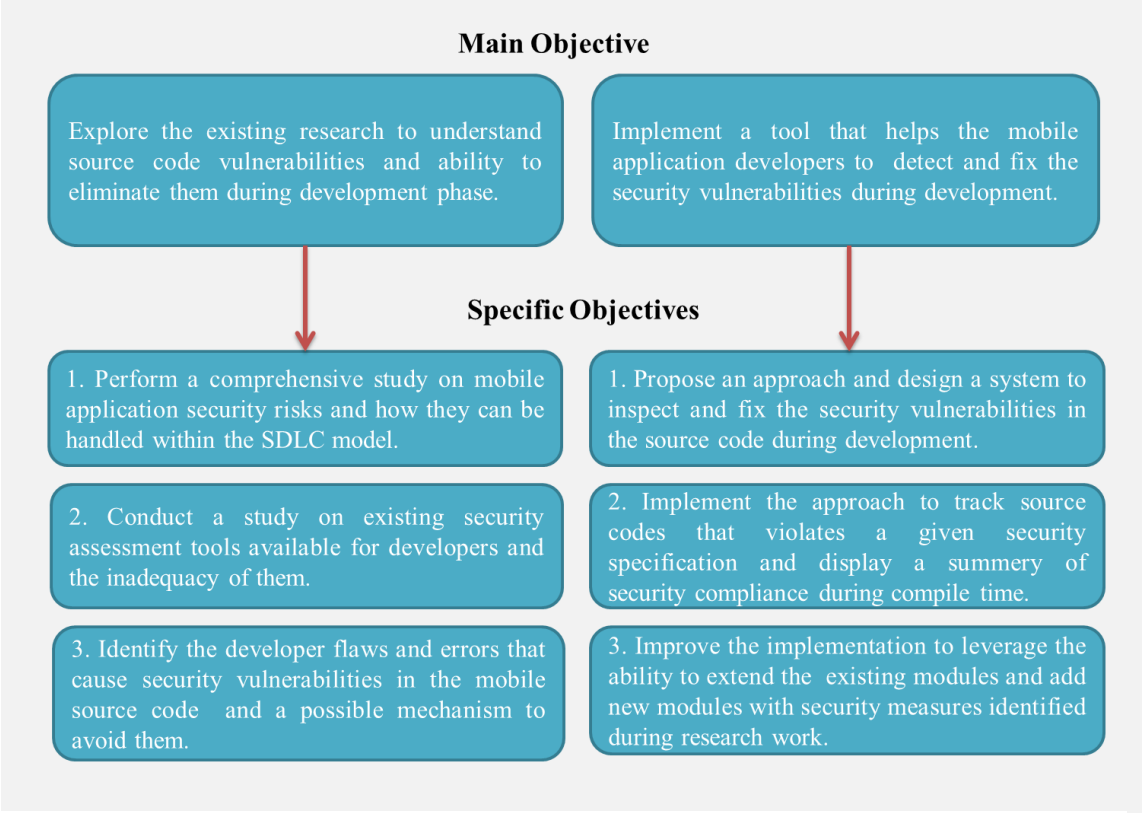

Figure 1.1: Objectives of the research project

As shown in Figure 1.1, the main objective of this research project is divided into two parts as research component and implementation work. One of the main objectives of this project is to explore the existing research work to understand the risks that are associated with mobile application development, developer issues that cause the mobile application source code vulnerable to security threats and for how far the existing assessment tools are capable of eliminating security threats within the development phase.

The other main objective is to implement a build tool that is cable of analyzing the source code for a predefined set of security vulnerabilities and help the developers to fix them during the implementation stage before the application is built into an executable file. The specific objectives that are expected to accomplish in order to achieve the main objective are stated in Figure 1.1.

General Objectives

- To explore the existing research to understand source code vulnerabilities and ability to eliminate them during development phase
- To implement a tool that helps the mobile application developers to detect and fix the security vulnerabilities during development phase

Specific Objectives

- To perform a comprehensive study on mobile application security risks and how they can be handled within the SDLC model
- To conduct a study on existing security assessment tools available for developers and the inadequacy of them
- To identify the developer flaws and errors which cause security vulnerabilities in the mobile source code and a possible mechanism to avoid them
- To propose an approach and design a system to inspect and fix the security vulnerabilities in the source code during development
- To implement the approach to track source codes that violates a given security specification and display a summary of security compliance during compile time
- To improve the implementation to leverage the ability to extend the existing modules and add new modules with security measures identified during research work

CHAPTER 2
LITERATURE REVIEW

2.1 Secure Software Life Cycle (SSLC) Model

Secure software development model is a research outcome where an iterative lifecycle for secure software development is introduced to mitigate security issues. SSLC consists of the phases of the traditional waterfall model with enhanced security features. This approach, as it is or with little amendments is being utilized in most of the enterprise level software and mobile application development companies. Figure 2.1 shown below illustrates the iterative life cycle for secure software development [6].

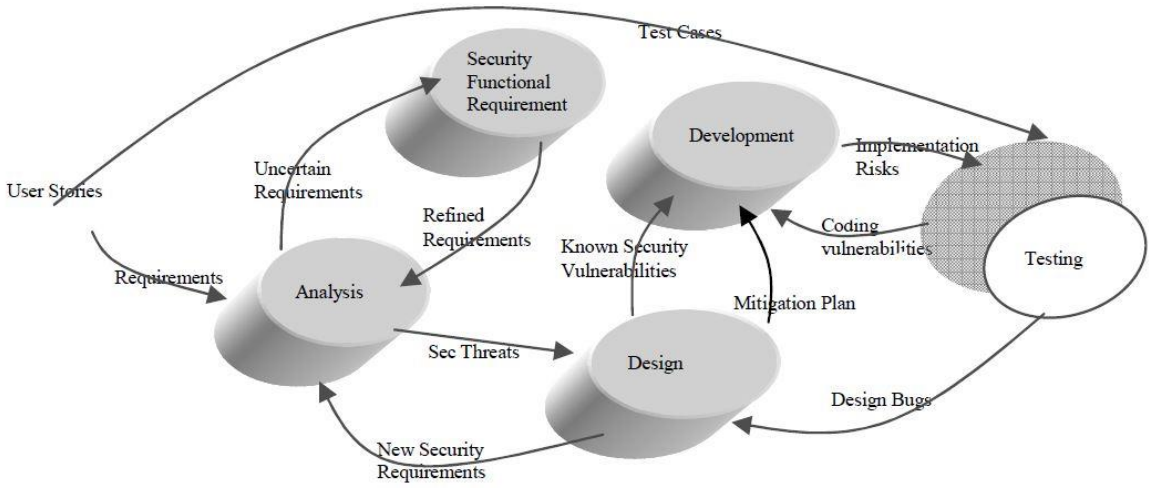

Figure 2.1: Iterative life cycle for secure software [6]

In the first phase of the given SSLC model which is the requirement phase, security requirements are elicited and derived using different methods. The security engineers will make use of user stories, functional and non-functional security requirements and -abuse cases to perform these tasks. In analysis phase, these security requirements are refined by a security functional requirements (SFR) module. All the requirements are mapped into functionalities during the design phase. The application architect has to develop a threat model to identify the potential threats, security vulnerabilities and their countermeasures. The two main activities of this phase are analyzing vulnerabilities and

identifying entry and exit points of all potential threats. As per the research outcome, below (in Table 2.1) are the common vulnerability areas that need to be addressed [6].

Table 2.1: Common vulnerability areas and types

<table>
<tr><th>Vulnerability Area</th><th colspan="2">Vulnerability Types</th></tr>
<tr><td>Operating system(OS)</td><td colspan="2">Buffer overflow(Stack, Heap), Null pointers, OS Resources deadlock, Exceptions etc</td></tr>
<tr><td>Communication</td><td colspan="2">Non repudiation of origin, Non repudiation of receipt etc</td></tr>
<tr><td>Database/User Data</td><td colspan="2">Invalid Data types, SQL injection, Cross Site Scripting, Rollback, Data integrity etc</td></tr>
<tr><td>Cryptography</td><td colspan="2">Key Management, Cryptographic operation, etc</td></tr>
<tr><td rowspan="2">Access Control</td><td>Authentication</td><td rowspan="2">Access control policy, data authentication, information flow control policy etc</td></tr>
<tr><td>Authorization</td></tr>
<tr><td>Privacy</td><td colspan="2">Privileges, Anonymity, pseudo anonymity etc</td></tr>
<tr><td>Programming</td><td colspan="2">Exception etc</td></tr>
</table>

Source: [6]

Vulnerability areas listed in Table 2.1 can be considered as security use cases for a given project. The countermeasures of these use cases have to be figured out afterwards. The system should be ready for implementation only after all the security attacks and vulnerabilities are identified. The implementation phase is the most challenging phase where security vulnerabilities and their countermeasures [7] are need to be addressed while coding and software configurations. However, in SSLC there was no attention paid to introduce a proactive mechanism to prevent vulnerabilities or security issues during the development phase. Hence, this model operates in a way where the identified vulnerabilities are manually addressed by developers and send them for testing. This does not guarantee that all the vulnerabilities which are identified during previous phases are addressed during the development phase. Consequently, this model will have several iterations between development and testing phases which will increase the

project cost due to repeated development and testing efforts. The next phase, i.e. security testing plays an important role in identifying security issues before the application is released to the end users. Hence, apart from the usual functional testing, the testers need to carry out “risk based testing” to ensure security vulnerability free software is delivered to the customer. “Penetration” and “fuzz” testing are the two main testing approaches used during “risk testing”. If any security flows are identified, this model is performed iteratively to get rid of them. Once all possible software security vulnerabilities are addressed the software will be ready for deployment [6].

2.2 Software Security Checklist (SSC)

As security assurances are integrated to the software development life cycle process to improve the software security, a security checklist (SSC) which is an instrumental guide that helps the software development teams to integrate security into the software development life cycle was presented. Integrating security is a critical task. Therefore it should be carried out as a formal approach within the software development life cycle. This can be done by incorporating software security checklists and security assessment tools into the software development life cycle process [7].

The steps to implement a good SSC process are as below [7]

1. Analyze security risks
2. Identify requirements and risks attached to them
3. Use a SSC instrument/tool in all phases of the development life cycle
4. Derive traceable and also verifiable security requirements for each phase
5. Asses security in each phase using the SSC

In Section 2.4, these tools and instruments are discussed in detail.

The most critical areas of all phases of the software development life cycle can be handled using a properly designed SSC. There are two main types of SSCs used in the software development phase.

- Type 1 – SSC for application development and maintenance
- Type 2 – SSC to verify external releases of the application

Maintaining both these checklists is highly important to the developers as it is them who are responsible for application development and maintenance as well as application releases.

Table 2.2 lists down several critical items that can be considered in generating a SSC for software development life cycle. This is just one example. It can be modified, extended or replaced to cover any security risk or vulnerability identified in the requirement analysis or the design phase [7]. The mobile security risks and vulnerabilities in mobile applications are further discussed in the Section 2.3.

Table 2.2: Items for potential consideration and inclusion in a SSC

1	Introduce a walkthrough, security audit review or a formal security review in every phase of the software life cycle development.
2	Establish security metrics during the software life cycle and a trace matrix for security requirements.
3	Determine stakeholders, and elicit and specify associated security requirements for each stakeholder
4	Determine context and potential usage of software product along with the operating environment and specify requisite security requirements.
5	Make available to programmers, developers, reviewers and test teams the vulnerabilities and potential exposures associated with programming languages and operating systems before the architectural design phase.
6	Set up security parameters for access to services such as *ftp* service where anonymous ftp is allowed but with write only and no read or list to the incoming directory and read only for outgoing directory
7	Check for sources of software security risks such as inconsistencies in requirements and in design, reusable programs and other shrink-wrap software. Use of requirements tools, modeling tools, etc. can aid in this area.
8	Avoid the use of unsafe routines such as sprintf(), strcpy/cat(), *gets* and *fgets* in coding.
9	Check the security of any middleware in the program.
10	Check for architectural-specific vulnerabilities and how data flows through the code.
11	Check for implementation-specific vulnerabilities such as Race Conditions, randomness problems and buffer overflows.
12	DO NOT allow programmer backdoors or unauthorized access paths that bypass security mechanisms.
13	Avoid storing secrets like passwords in the code or use weak encryption schemes
14	Identify all points in the source code where the program takes input from users.
15	Identify all points in the source code where the program takes input from another program or un-trusted source.
16	Investigate all sources from which input can enter the program such as GUI, network reads, etc.
17	Check API (Application Program Interfaces) calls to security modules or interfaces.
18	Investigate secure connections. Verify that they actually are secure and connect as indicated to the systems to which they are intended to connect.
19	Investigate software built-in extensibility features.
20	Review software complexity and look for alternatives to reduce the complexity.
21	Investigate the security of the data when passed from application servers to databases.
22	Avoid default or other improper configurations that may open the door to attackers.
23	Default to "highest security" needed, and require validation and approval for deviations.
24	Establish tools to be used for various stages of the life cycle that will be used for assessing security.
25	Perform security testing for unit and system integration.
26	Potentially, establish a security risk rating criteria and document the rating of the software product within the organization. Using a risk assessment tool can benefit this area.

Source: [7]

Table 2.3 provides an example of an initial start of a security checklist for the software that is developed for release external to the organizational environment.

Table 2.3: Example of a security checklist for the external release of Software

1.0	Does the software include IP addresses and subnet ranges? 1.1 If yes, are these IP addresses sensitive? 1.2 Can these addresses be used to gain information that may pose a risk to the organization?
2.0	Does the software/program include Host names? 2.1 If yes, are these host names sensitive? 2.2 Does the release of these host names pose a risk to the organization?
3.0	Are there any settings that can be exploited? 3.1 If yes, can any of these settings be modified or deleted? 3.2 If settings can not be modified or deleted, would they pose a risk to the organization? 3.3 If settings can be modified or deleted, would they pose a risk to the organization?
4.0	Is there any non-sensitive information in the software that can be used to probe secrets? 4.1 If yes, can non-sensitive information be manipulated to expose sensitive information? 4.2 Can non-sensitive data be altered and modified so as to pose risk?
5.0	Is there any Material that might expose company information such as Customer lists? 5.1 If yes, are Customers lists protected under a privacy policy? 5.2 Does the release of Customers lists pose a risk? 5.3 Does the release of Customers lists do harm to the customers?
6.0	Is any of the data restricted? 6.1 If yes, is the data controlled by security mechanisms such as RBAC? 6.2 If yes, are their security restrictions on the transfer of restricted data? 6.3 Is the restricted data transmitted over open networks? 6.3.1 Is the restricted data encrypted before transmission?
7.0	. . .

Source: [7]

Given below is another set of security vulnerability diagnostic items presented by a joint effort of Computer Science Departments of Konkuk and Shamyook Universities. These security vulnerability diagnostic items are generated based on a mobile application security review in order to prevent security accidents that can occur in a mobile service environment. These checklists are based on analyzed data collected from Android applications [14].

1. Permission Management – Access permissions and privileges of other applications such as data sharing and management
2. Input data validation - Validation of input data through the users
3. Important file management – Safety check of important file checks

4. External data transfer management – Important data encryption communicating with external data
5. Component management – Abuse check of the used components
6. Security program check – Data explore and safety check in the program code
7. Data use policy management – Use of personal information and violation of the mobile platform, the security model and user authentication
8. Safety management for the open module – The public availability of the safety check for the open module
9. DB data management – Maintaining the safety of the database data verification

Following all the aforementioned checklist items is not a sole responsibility of the developers as the functions of these phases are performed by the other roles as well. However the developers need to ensure the items belong to development process are attended and checked. In the Section 2.5, the developer responsibility to handle security vulnerabilities in the source code is further discussed.

2.3 Mobile Application Security Risks

To move on to "mobile application security" which is the core topic of this research project, it is necessary to understand what are the security risks associated with mobile application development.

As identified by different authorities and organizations that has performed continuous research and conducted studies on this topic, there are several security risks and vulnerabilities in mobile applications that expose the mobile application users to extreme security threats. OWASP and Veracode are two such organizations that had conducted research on this area for a quite a long time. Below are their latest findings.

The top 10 mobile risks published by OWASP in Mobile Security Project

The Open Web Application Security Project (OWASP) is a combined group of resources formed to provide support and assist developers and software security professionals to build and maintain secure mobile applications. The "Top 10 Mobile Application Risks" is a descriptive list of mobile application behaviors that would impact application user's security. This is published by OWASP to educate mobile application developers and security professionals [12].

1. Insecure data storage

   Allowing data to be unprotected on cloud based data storage or storing sensitive data in device storage. The technical reasons behind this would be, not encrypting stored sensitive data, caching user information which is not intended to store, not setting correct file access permissions or not applying the best practices recommended for the application platform. Insecure data storage will drive applications to be security noncompliance and lead to privacy violations by exposing sensitive information.

2. Weak server side controls

   Data confidentiality and integrity can be compromised by failing to maintain a proper security mechanism during application updates/patches, changing security configurations or default setting/accounts, disabling or enabling backend services.

3. Insufficient transport layer protection

   The most common client-server communication protocol used in mobile applications is HTTP which transfers all the information in plain text. Even if HTTPS provides transport layer security, in case of a certificate validation error is ignored or plain text communication is reverted after a failure, it will lead to revealing data to man-in-the middle attacks.

4. Client side injection

Web based and hybrid mobile applications which use lot of web technologies are highly exposed to HTML, XSS and SQL injection attacks. Apart from that, client side injection attacks are executed through phone dialer, SMS application and in-app payment modules.

5. Poor authentication and authorization

Mobile applications without strong authentication and authorization mechanisms are highly unsafe to use. For the ease of development it is not recommended to rely on device identifiers or Universally Unique ID (UUID) for security. Using them may lead to broken authentication and unauthorized access issues.

6. Improper session handling

It is not recommended to use device identifiers such as session ID or set a long expiration time for sessions in mobile applications. These could lead to unauthorized access as well as privilege escalation.

7. Security decisions based on untrusted inputs

Applications that make security decisions based on user inputs are easy targets for malware and client side injection attacks. The possible attacks would be privilege escalation, data exfiltration and paid resources consumption. Abuse URL schemes in iOS and abuse intents in Android applications are examples for possible security manipulations.

8. Side channel data leakage

Flaws in the source code or not disabling insecure OS features lead to sensitive data to be stored in web cache or temp directories and traceable in global OS logs. These flaws open doors to malware and attackers to easily access sensitive data.

9. Broken cryptography

This originates from not following secure development practices and lack of knowledge. Bad development practices such as using custom cryptographic algorithms instead of standard ones, misunderstanding obfuscation and encoding and use encryption instead, keeping hardcoded cryptographic keys in the source code will impact confidentiality of data and result in privilege escalation.

10. Sensitive information disclosure

The most common way of sensitive data disclosure is developer mistakes. Developers tend to hardcode sensitive data in the source code with the intention of debugging or for temporary use but forget to remove them. There are instances where developers do this due to lack of knowledge and best practices. These sensitive data are typically login credentials, access tokens, shared keys and sensitive business logics targeted to acquire by attackers.

The top 10 mobile security risks in mobile app published by Veracode

Veracode breaks down mobile application security risks into two main categories. The first category “Malicious Functionality”, lists down behavioral security issues of mobile applications which are unsafe and not recommended for mobile application users. The second category “Code Vulnerabilities”, lists down security vulnerabilities caused by errors in mobile application design and implementation which expose sensitive data to outside world allowing unauthorized access [13].

A. Malicious Functionality

1. Activity monitoring and data retrieval

The ability gained by the attackers to intercept information and data by monitoring them in real time while they are being generated in the device.
Example:

- Generate emails to a 3rd party address when an email is sent from the device
- Enable microphone recording through a malware to listen to phone calls
- Retrieve emails, contact list or any other data stored in the device

2. Unauthorized dialing, SMS and payments

The ability the attackers gain to use a compromised device for unauthorized monetize usage.
Examples:

- Install a Trojan app which executes premium dialing functionality to runs up a mobile user's phone bill and make the carriers to distribute the collected money to attackers.
- Install a malware in a mobile device to use it to purchase real or virtual items and include that cost to the mobile user's phone bill.
- Gain access to unauthorized SMS text messaging and spread worms to user's contact list by including a link to download and install the worm.

3. Unauthorized network access

Mobile device communication can be compromised using malicious applications designed to intercept, retrieve and send data to attackers. A well designed malicious program can obtain direct commands to instantly turn on device microphone or access a data file at a given time. Bluetooth, SMS, Email, TCP/UDP sockets, HTTP

GET/POST and DNS exfiltration are examples for communication channels that can be compromised for exfiltration and command control.

4. UI Impersonation

These are more like phishing attacks where the user is tricked to submit sensitive data such as credentials by clicking a link in the browser. In mobile devices this could be presented as a native mobile UI which is a proxy to a native web application or a malicious application with UI pops ups which impersonate a native UI of a genuine mobile application.

5. System modification

This is often referred to as rootkit behavior where the system or device configurations are modified or changed by malicious applications to hide their existence compromising the devices to more attacks. Applications that attempt to modify device's proxy configuration and Access Point Name (APN) belong to this category of threats.

6. Logic or time bomb

This is a backdoor technique used by the malicious applications to trigger activities at a given time or at a specific event. For an example, executing a malicious activity during certain hours of the day or on a specific day of the week or when making a phone call or on receiving an email or a SMS from a specific person.

B. Code Vulnerabilities

1. Sensitive data leakage

Sensitive data leakages can be caused either by side channels or developer mistakes. An erroneous source code may expose user's personal and sensitive data to unauthorized third patties causing data privacy issues.

2. Unsafe sensitive data storage

Sensitive data such as PIN numbers, credit card details and passwords for online accounts are often stored inside mobile applications. In order to avoid unauthorized retrieval of these sensitive data, it is highly recommended to use a strong cryptography to keep them encrypted. The security risk is even higher if such data are stored in removal Secure Digital (SD) cards without encrypting them.

3. Unsafe sensitive data transmission

Sensitive data needs to be encrypted during transmission. As mobile devices often use public Wi-Fi for communication, the data in transit is at high risk of being retrieved by third party attackers. SSL is one proven mechanism to secure sensitive data during transmission.

4. Hardcoded passwords/keys

Developers often keep keys and passwords hardcoded in the source code for easier implementation, debugging and maintenance purposes. Access to these hardcoded values can be easily obtained through reverse engineering of the package file or other methods. This will lead to data privacy issues and impact the security compliance status of the application.

Mobile application vulnerabilities identified by DHS

According to a study conducted by DHS (The Department of Homeland Security) on Mobile Device Security in 2017 [15], the vulnerabilities identified in mobile applications are as listed below.

1. Third party applications running in jail-broken or rooted devices

   There are certain applications which cannot be run unless the mobile device is rooted or jailbroken. Users without much knowledge root their devices with the intention to install and run such applications not knowing they compromise their devices to security threats by degrading the security state of the device. In such state, it is quite easy for a malicious application to perform unauthorized activities within the device.

2. Insecure network communication

   An unencrypted network communication between the mobile applications and the remote servers let attackers to eavesdrop on the network connection and obtain sensitive data in transit. Attackers can also modify the data in transit to deliver compromised information to the other end. When the identity of the server is not correctly authenticate during connection establishment that will also open space for man-in-the-middle attacks.

3. Sharing data with untrusted apps

   There can be applications running in the device that will sync user's sensitive data to external applications or sources (for an example "Dropbox") without the knowledge of the user. Such activities not only lead to data loses but also violates user's data privacy by letting attackers to get access to users sensitive data and put him at risk.

4. Files stored with insecure file permissions or in an unprotected location

When files are stored, it is required to set the file permission with correct access privileges and store them in a protected location. Failing to follow these guidelines when storing files which include unencrypted sensitive data will lead to data leakages and unauthorized data access.

5. Sensitive data and logs written to files

It has been discovered that logs written to plain text files through mobile applications built using Android and iOS platforms can be accessed by the attackers. Apart from the system logs, the console logs maintain during coding for debugging and documentation purpose should also be removed before application is released as the content written to the logs may contain sensitive data.

6. Web browser vulnerabilities

Applications that run on web browsers are used as entry points to obtain access to mobile devices by the attackers. Latest versions of mobile application development platforms such as Android and iOS have made improvements to their security architecture designs to discourage this kind of attacks. Mobile device users are recommended to use stable and secure browsers. In case of web browser vulnerability it is advised to back up their data to external sources and reset the devices to default factory settings.

7. Vulnerabilities in third-party libraries

Reusable third party libraries, plugins and other software components are often utilized during mobile application development as reusable components and modules usually make the development process more efficient. If any third party component used is security compromised or flawed with vulnerabilities that will impact the

security status of the entire application as it open doors to security threats putting many users at risk.

8. Cryptographic Vulnerabilities

Failing to protect sensitive data with proper use of cryptography leads to this kind of vulnerabilities. There are many cases where cryptographic algorithms are used to protect data but either with wrong implementations or the algorithms are customized and not up to the standard of industry recommended cryptography. Therefore both failing to use and using incorrect cryptographic algorithms make the applications security vulnerable and allow unauthorized access to data.

2.4 Security Assessment Tools and Instruments Used in SDLC

Security assessment tools are valuable and useful in producing secure software. There are number of tools currently available for security assessment and testing of mobile applications throughout the software development life cycle. Security testing is usually performed either in QA or UAT phase of the SDLC. In mobile app security testing, the security assessment tools are categorized into three major types of testing tools as: static, dynamic and forensic.

1. Static testing tools

Static testing tools are typically used to test security vulnerabilities in the source code, application package or binary files. These tools track the security vulnerabilities that could arise when the source code is running on the device. For an example, a static testing tool would trace an invalid buffer handling or an issue with the dataflow in the mobile application source code.

2. Dynamic testing tools

Dynamic testing tools are used to observe the behaviors of an application running in a simulated environment similar to the actual environment and track security vulnerabilities in the system. Proxies are the most commonly used dynamic analysis tools used for mobile application security testing. Proxies can be used to monitor the communication between the application and the remote servers. Using proxy tools, the communication protocols can be reverse engineered to craft malicious messages which are not possibly generated by a genuine mobile client. Therefore, using such dynamic testing tools, potential server side attacks can be simulated and tested.

3. Forensic testing tools

Forensic testing tools are used to examine the artifacts such as source code, external files and third party plugins which are left behind once the application is compiled and run. These tools typically trace hardcoded keys and passwords in the source code, sensitive data stored in configuration files, databases and web browser cache. Sophisticated forensic tools are even able to check if the access controls of the operating system in which the mobile application is deployed are correctly enabled on the components of the mobile application under testing.

Mobile applications usually have complicated threat models which need to be examined from different aspects during security testing. A comprehensive testing process should ideally use set of tools from combination of all 3 categories [17]. Therefore, most of the existing security assessment tools consist of components that cover static, dynamic and forensic testing to ensure that mobile applications are tested in every security aspect.

Table 2.4 contains a list of freely available, open source and inbuilt security assessments tools available for mobile platforms. Even though these tools cover most aspect of security testing, none of them has the ability to automatically fix the security

vulnerabilities that are found in the source code. Most of them do not have a flexible report generation mechanism as well.

Table 2.4: Existing security assessment tools

Tool/Instrument	Type	Supported Mobile Platform	Usage
Clang Static Analyzer	Static	iOS	A static analysis tool for C, C++ and Objective-C programs. This is used to test for certain quality and security errors in iOS-based applications. This can be run from both command line and inside Apple's XCode development environment.
'otool' command by Xcode	Static	iOS	This XCode-provided "otool" command can be used to extract information from iOS application binaries that can be used in support of security analysis.
FindBugs along with DeDexer and dex2jar	Static	Android	DeDexer can be used to generate DEX assembly code from an Android DEX application binary. dex2jar can be used to convert DEX application binaries to standard Java JAR files. FindBugs is a Standard Java analysis tools that can be used to analyze these JARs.
JD-GUI	Static	Android	A Java decompiler that converts the Java byte code back into Java source code which helps to review or scan the code for vulnerabilities.
OWASP Zed Attack Proxy.	Dynamic	Android and iOS	A framework that provides a real environment for mobile testing infrastructure and mobile devices. It supports the installation of additional tools and platform for penetration testing. The detection of system vulnerabilities can be performed automatically.
Android Debug Bridge	Forensic	Android	This is a command line tool that comes with Android development KIT and is provides some commands that helps to explore the android file system and system data.
iPad File Explorer	Forensic	iOS	This allows browsing files structure on iOS device. iPad file explorer can list out application data and media files in different views. It can also access storage and file system of rooted or jail-broke devices.
The SQLite database engine	Forensic	Android	SQLite 3 command line program allows to query the databases created by the android application and stored in the device memory. This can be used to reveal sensitive information such as password, PINs hashed or stored in clear text.
Santoku	Combination of all 3	Android and iOS	This is a virtual machine consists of a set of open source security testing tools for mobile applications. It also has malware analysis, data recovery and forensic testing tools.
MobSF - Mobile Security Framework	Static and Dynamic	Android and iOS	This is an automated framework designed for penetration testing. This can be used in both Android and iOS mobile platforms for static analysis, dynamic analysis and web API testing.
Mitmproxy	Dynamic	Android and iOS	This is a proxy which interacts as a man-in-the middle attacker for HTTP and HTTPS connections between mobile applications and remote servers. It has a console interface that allows intercepting and modifying network requests and responses.
Drozer	Dynamic	Android	This is capable of identifying security vulnerabilities in Android applications and devices by discovering and interacting with the attack surfaces exposed to outside parties by the applications. This tool removes the need to install test scripts on the device by running a dynamic java code on the device for security testing.
Frida	Dynamic	Android and iOS	This is a toolkit that injects JavaScript code snippets to native mobile applications to trace security vulnerabilities in the application without the source code.
Radare	Combination of all 3	Android and iOS	This is a framework that reverse engineers mobile application binary files to inspect and analyze security vulnerabilities. This also has capability to debug application code with local and remote debuggers and perform forensic testing on data flow and file system.
QARK	Static	Android	This testing tool traces security vulnerabilities in Android application source code and its package (APK) file.
Kiuwan	Static	Android	This is an end-to-end analytical platform for static source code analysis and automated code review. It can detect defects in the source code, trace security issues and manage security risks with its inbuilt application governance feature and enhanced life cycle.
Amandroid	Static	Android	This is designed to inspect and analyze security vulnerabilities in the data flow between the internal components of Android applications.

2.5 Causes and Elimination of Source Code Vulnerability

The main reasons that cause security vulnerabilities in mobile applications are identified to be the mistakes or the errors in the source code and not following secure coding practices when applications are being developed. The security risks are caused when these security vulnerabilities are exploited to compromise privacy and integrity of user's data. Security risks associated with application source code can be avoided if the source code is properly and thoroughly reviewed during application development prior to production release. Even though the risks caused by erroneous coding can be diminished up to a certain level by the mobile device architecture, some security vulnerabilities which are injected during coding or application development may still be unrecognized or unnoticed before the application is released to the end users. Even if such security vulnerabilities are identified, the application will remain to be a risk to the end users unless the application is updated with the security fixes or removed from the marketplace. As identified by the studies, the key reasons the security vulnerabilities are introduced during coding or software development phase are as below [7].

- Developer mistakes and carelessness that result defective source codes left to be handled by the compilers to diagnose errors
- Low reliability of the mobile applications due to constant demand to incorporate new tentative features even when they are at the edge of the production phase
- Lack of proper software engineer training and skilled developers
- Lack of developer supportive resources such as source code analyzers that help developers to trace security vulnerabilities during development

Following secure development guidelines and best practices recommended by mobile application development platforms during coding ideally helps to reduce or completely eliminate the known security vulnerabilities. There is a set of best practices recommended for mobile application developers that should be applied during coding and there is another set which should be followed during the application maintenance.

Developer training and awareness of secure coding guidelines

In order to build secure mobile applications, it is highly important to train developers and make them aware of secure coding guidelines and best practices. It is recommended to conduct a developer training on secure application development and common programming mistakes and errors that could impose security vulnerabilities in the source code prior to development phase. It is a must for a mobile application development team to be aware of secure coding guidelines and best practices published by Apple for iOS development [22] and Google for Android development [20]. Knowledge on mobile application risks and vulnerabilities presented by OWASP would also be an advantage [12]. It is highly recommended for mobile application developers to be aware of the following secure development guidelines at minimum to ensure secure coding [19].

1. Perform secure logging and error handling

   During development, developers usually maintain commented codes and write logs for different purposes. The activities stated below which are related to maintaining logs in mobile applications can expose sensitive information to external parties. Therefore it is recommended to avoid following activities during coding [19].

   - Keeping logs in the global log
   - Keeping logs as commented codes for debugging purposes
   - Handling exceptions badly in the source code

2. Follow the principle of least privilege

   To ensure security, it is recommended to sandbox and isolate the mobile application. To accomplish this, the developers have to implement the permission model of the mobile operating system correctly by following "the principle of least privilege" [19]. When requesting for user's permissions, the best would be to request the least amount of permissions required to run the application. It helps restricting the access to unwanted sensitive permissions and thereby avoids misusing them. Simply, a

mobile application should not request for any permission which is not needed for application functionality [21].

3. Validate input data

Not performing proper input validations is identified as one of the common reasons for security issues in mobile applications. Android mobile application development platform provides a set of countermeasures that helps mitigating the security issues involved with input validations. Developers are recommended to use this as s secure coding practice [21]. Another important practice that should be enforced in mobile application development is to ensure that all client side input validations are duplicated at the server end as well. Implementing "OWASP's Enterprise Security API" which is a security control for input validations is also a recommend method to avoid input validation security issues [19].

Input validations need to be done when data is being retrieved from any external storage as these external sources cannot be trusted. It is recommended not to store class files or executables on an external storage or not to retrieve any executable from an external source without signing them or verifying cryptographically before dynamic loading [21]. Any data travelled through an external network or retrieved through an inter process communication (IPC) is potentially harmful. The most commonly expected security issues are buffer overflow, off-by-one error (OBOE) and use after free error. The best ways to prevent these issues are to handle pointers and manage buffers appropriately [21].

SQL and JavaScript injections are also associated with input validation issues. SQL injections could cause security issues when mobile applications use SQL queries to submit data to databases or content providers. Most of the issues related to SQL injections can be avoided by using parameterized queries and restricting permissions to write-only or read-only [21].

4. Implement secure data storage.

As a best practice, sensitive data is not recommended to be stored in SD cards or other external storage. The files stored in SD cards can be accessed globally to read and write data hence, other applications can easily modify data stored in SD cards. External storage can also be physically detached by the users causing data loses. On the other hand, files stored on internal storage can only be accessed by the applications with controlled access permissions. This is a platform specific implementation provided by Android to secure application data [21]. When encrypting sensitive data, it is not recommended to use custom encryption algorithms but standard ones with strong key values [19].

5. Avoid insecure mobile OS features

There are some features provided by mobile operating systems by default which are identified to be insecure in certain application contexts. Given below are few such features which can affect mobile application security. They should be disabled in mobile applications which hold lots of sensitive information [19].

- cut-copy-paste
- auto-completion
- baking up application data
- installation on rooted devices

6. Encrypt data in transit

It is a must to take appropriate actions to protect data being transferred from mobile applications to the backend servers. A special attention should be given to protect data that carry authentications tokens, sessions IDs and sensitive user details. Unsecure connections such as public Wi-Fi networks can easily be interfered using latest hacking techniques. To avoid this security issue, it is recommended to use

SSL/TSL connections for communication between mobile applications and remote servers [23].

7. Encrypt sensitive user data

Developers must ensure to encrypt application data which contain users' sensitive information such as login credentials, contact details, passwords and PIN numbers. Neglecting to encrypt sensitive data used in mobile applications not only put the users but also the developers at risk by exposing data to unauthorized access and data breaches [23].

8. Protect user sensitive data

There are some mobile applications that require to access lot of sensitive user data such as login credentials, PIN numbers and credit card details. The best approach to protect these data is to avoid storing them in any persistent storage and minimize the occurrences of transmitting them to remote servers using API calls. As a best practice, it is recommended to use hash maps or nonreversible form of data during application logic implementation [21].

9. Protect user's application data

When a client side session is expired or logged out, the same session should be simultaneously invalidate from the back-end as well. As mobile applications that retrieve, collect and store sensitive data are targets for phishing attacks, it is recommended to use one-time application specific password or two-factor authentication (2FA) via SMS, phone call or email [23].

10. Handle authentication with care

In order to make phishing attacks ineffective it is recommended to minimize the number of times a mobile application requests for user credentials. Best practice is to use a token for authorization and refresh it as requires. Mobile application should request for user credentials only once at the initial authentication. Afterwards, a service specific short-lived authentication token should be used to access different services until the user logs out of the application. This method avoids having to keep the user credentials saved on the device and mitigates phishing attacks [21].

11. Use explicit intents over implicit intents

In Android development, activities and broadcast receivers mostly use intents for asynchronous inter-process communication (IPC). Based on the requirements, developers have to use one of the two methods from "sendBroadcast()" and "sendOrderedBroadcast()" or an explicit intent. For security-intensive mobile applications it is recommended to use explicit intents. Implicit intents should be avoided in such applications as the user is unaware of the services started and also it is not certain which services would respond to the intent. Such situations might lead to security hazards [21].

Use of security assessments tools that can assess application for vulnerabilities

There are free security assessment capabilities and features bundled into the mobile application development platforms. For an example, Android Studio comes with "Android Software Development Kit" (SDK) and "Android Lint" which helps developers to impose security best practices and assess security vulnerabilities during implementation. There are more sophisticated security tools which are integrated with intelligence and contain information about more up-to-date source code vulnerabilities [15].

2.6 Similar Work

The outcomes of several other research projects which have influenced on the idea of the presented approach are discussed in this section. These studies have influenced on shaping up the presented approach in several ways but they are more focused on securing overall mobile application design, architecture and the infrastructure and less focused on securing mobile application implementation.

A hierarchical framework model of mobile security

This research study presents a framework that guides investigation of security vulnerabilities in mobile applications using a systematic approach. It is presented as a hierarchical model with three security layers. The three layers are “Property Theory”, “Limited Targets” and “Classified Applications”.

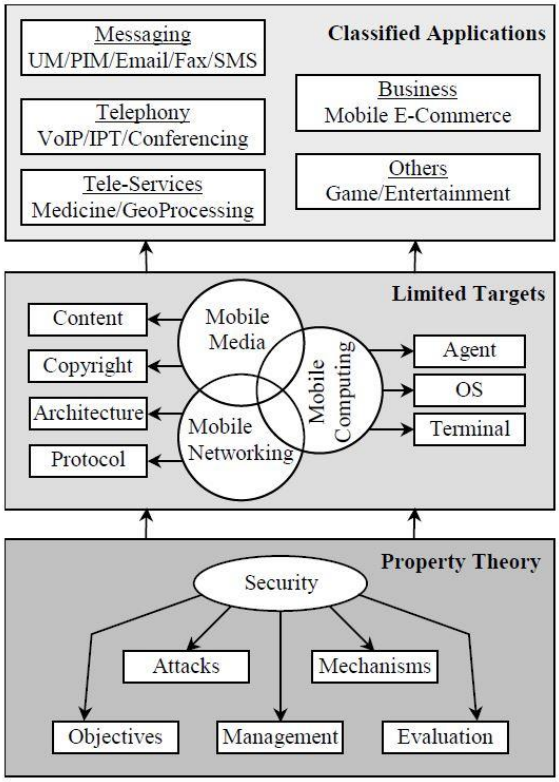

Figure 2.2: Hierarchical framework model of mobile security [24]

Property Theory focuses more on the technical area of security by categorizing the security into objective, attack, mechanism, management and evaluation. According the study, security objective should be to emphasis more on the Confidentiality, Integrity and Availability (CIA) triad in protecting data. Security attack discusses the possible intrusion orientations, source, target and methods. Security mechanisms presented by the study are encryption algorithms, access control, barring and filtering firewalls, security protocols, intrusion detection, scanning etc. Limited Targets Layer is focusing on the mobile network security in IP networks, 3G wireless networks and mobile software agents where host and agent protection is taken into discussion. Classified application layer is the other layer where application domain specific issues are discussed. There is always an additional set of security threats coming up based on the domain in which the application is operating in [24].

In conclusion, this study provides a set of guidelines to secure overall mobile system architecture whereas the presented approach is focused on eliminating source code vulnerabilities and enforce secure mobile application development phase.

An approach to secure mobile enterprise architectures

This study is conducted to demonstrate concepts that can be applied to provide overall security to the enterprise mobile application architecture. It presents a conceptual security solution constructed with the help of several technologies, security standards and system components. It proves that security measures associated with individual mobile application components do not cover the security of the entire application. However, this presents an approach targeting the developers to reduce security issues occur in development phase. The research has also identified that, the lack of time allocated for development phase and the pressure put on developers to release applications with novelty features during a short time frame have pushed the developers to neglect addressing security issues in mobile application source code. Also, developers

tend to rely on security testing performed during testing phase to identify security vulnerabilities and fix them. Hence, this approach provides a security infrastructure, combining Two-Factor or Multi Factor Authentication (TFA, MFA) and Mobile Virtual Private Network (Mobile VPN) to securely authenticate users to the system and protect cooperate data in transit and device storage. This approach is helpful to developers in frequent secure application publishing. The Figure 2.3 illustrates how this infrastructure is applied to invoice approval architecture [25].

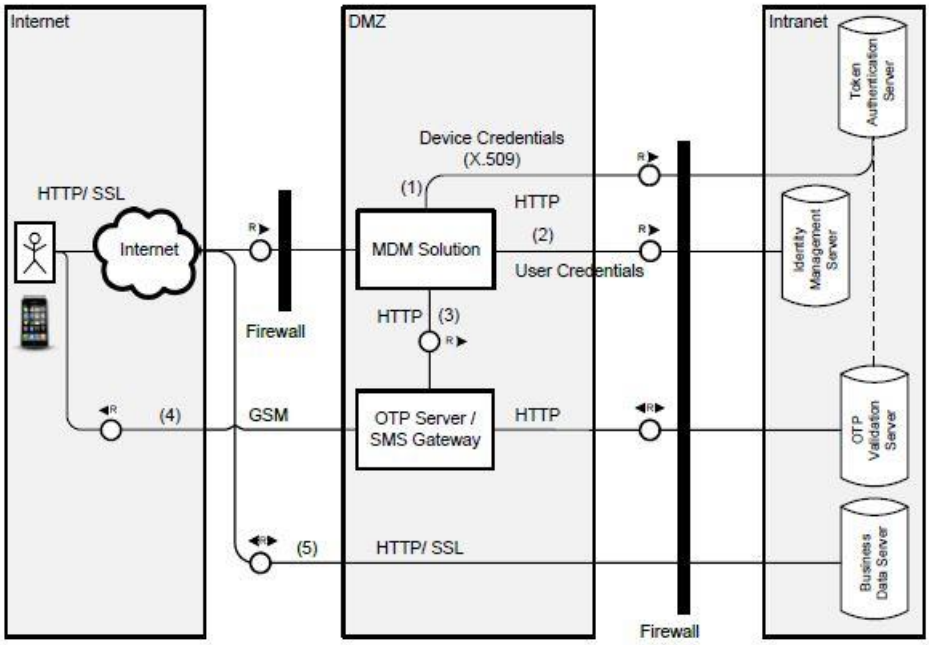

Figure 2.3: Invoice approval system in TFA/MDM/VPN infrastructure [25]

This study has paid some attention on secure mobile application development phase yet, it has been worked towards an infrastructure that helps developer to publish secure mobile application rapidly and not discussed any mechanism to secure mobile application source code.

Application security framework for mobile application development in enterprise setup

This paper introduces a security approach that can be imposed on mobile application layer which consequently reduces the risks in the enterprise. This framework is a result of an effort taken to come up with a set of mobile security standards in enterprise space. Enterprise or commercial mobile applications must be aligned with a fine set of security standards as such applications deal with lot of customer data and important business logics unlike the consumer or gaming applications [26]. The idea presented in this paper fairly influenced on the presented implementation as well. This research presents resolutions for the security vulnerabilities that could take place in the areas listed in Table 2.5.

Table 2.5: Areas of security vulnerabilities and proposed resolutions

Areas of focus	Security threats/ vulnerabilities	How the vulnerabilities are handle in the framework
Data protection	Data store in local storage	Conduct data audit to check the criticality of the data that remains in the device even for a short span of time
		Encrypt data stored in the local storage
		Perform application level granular check to ensure that local device database modification can only happen through the application code
	Cache usage	Ensure cache doesn't contain any critical information
	Data sharing	Applications should clearly partition the data within its boundary
	Data on transit	Authentication mechanism must be put in place to restrict movement of data from secure area to unsecure area
		Application should encrypt the message to be sent over the air
		Data communication channel should be secured via HTTPS instead of plain HTTP
		Data exchange that happens over SMS channel must also be encrypted and should happen over secure SMS protocol
Secure authentication	Session management	Maintain a session ID appended with additional unique information that identifies the device or user so that any unauthorized device cannot use the same password as pose as an authentic user to the application server
	Password management	Include TFA
		Use algorithms to ensure user set secure passwords
Intellectual property protection	Reverse engineering	Perform obfuscation to remove the debugging information as well as to mangled or replaced the object names within the byte code by meaningless entities, without hampering the way the application works
	Hardcoded critical information	Ensure that no critical information is getting hard-coded within the application code, including but not limited to crypto keys, user credentials, and other sensitive user information
Code vulnerability	Validation	Perform front and server end validation for source code with scripting
	Exception handling	Capture the stack trace and make it available to the development team for analysis of possible issues in the application
		Avoid showing the stack trace to the end user
		Handle exception with customized messaging to reduces the security exposure
	Using 3rd party libraries	Enlist the deprecated APIs when any source library, is used.
		Use processor based approach for other platforms where the deprecated APIs are not marked clearly.

Source: [26]

This paper does not present an architecture or implementation of how the system verifies these security vulnerabilities in the framework. Hence, it is not clear how these components collaborate or connect in the framework to ensure mobile application source code is secure. However, the resolutions presented in this research influenced on implementing verification methods for code vulnerabilities in the presented project (i.e. “buildsec”).

CHAPTER 3
METHODOLOGY

Based on the outcome of the research study, it is understood that the best instance where a practical resolution to the research problem can be applied at compile time of the mobile application. It is identified that manifest file and the java source code are the two main areas where developers introduce code vulnerabilities android mobile platform. Hence, the "buildsec" library is implemented to inspect the manifest and the source code, track the source code vulnerabilities and fix them based on a given set of security criteria to ensure the mobile application is security compliance.

3.1 System Architecture and Design

The Figure 3.1 illustrates how the "buildsec" library interacts with other components in android mobile platform.

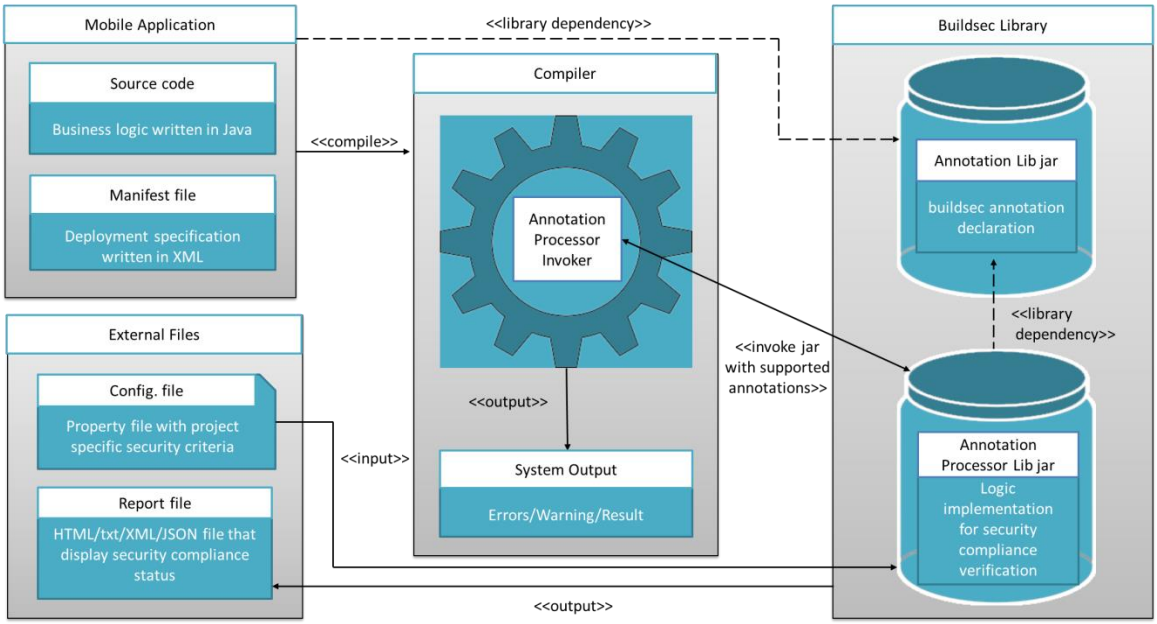

Figure 3.1: Design diagram of buildsec system architecture

Config File

This is the property file where the code level security criteria based on the project's security requirements are defined as key-value pairs. The compliance status of the application that is output form the "buildsec" library is based on the values assign for each security criterion defined in the "config file". This file also includes configuration settings to disable security verification of the mobile application at different levels.

The "buildsec" library consists of two individual java archives "Annotation Library" (buildSec.jar) and "Annotation Processor" (buildSecProcessor.jar).

Annotation Library

This is where the "buildsec" annotation is declared. This annotation needs to be used in the source code where security compliance needs to be verified.

Annotation Processor

This sub-library contains all the modules belong to "buildsec" where the logic for each security verification is written. Security verification logics can be written in multiple classes and methods within this jar. Hence, this library can be enhanced to support more security criteria verifications by adding more methods or enhancing the existing modules. Currently, "buildSecProcessor.jar" provides security verification for below security criteria.

- Uses permission
- Debug logs
- Auto backup

This jar reads the “config file” to retrieve the security standard and settings declared in it and verify the source code based on that.

Compiler

Compiler invokes the “Annotation Processor” when the mobile application is being compiled. “Buildsec” library executes the following tasks during compilation.

- Reads “AndroidManifest.xml” file and creates a list of XML tags explicitly specified in it (e.g., uses permission).

- Looks for android class files (java source code) and inspects them for code vulnerabilities (e.g., methods which are using API functions with special permissions which are not declared in the manifest).

- Reads the “config file” and retrieves the security criteria that need to be verified in the source code.

- Fixes the source code vulnerabilities if any security violation is found and recompiles the source code until it becomes security compliance.

- Generates the report file in HTML to display the detected security vulnerabilities and compliance status of the source code.

Report File

This could be a HTML, txt, JSON or XML file which include the following information.

- A list of detected security vulnerabilities

- Security compliance status of the mobile application

Figure 3.2 shows the “Report file” that is currently generated from the “buildsec” library in HTML format.

Build Secure Compliance Report			
Project Name	DemoApp		
Compiled by	prasad	JDK	1.8.0_0152
Date/Time	December 16, 2017 12:24:13		
	Status	Config	Source
Users permissions	pass	enable	valid
Data backup	pass	false	false
Log Entries	pass	disable	unavailable
Security Status	Compliance		

Figure 3.2: Sample security compliance status report

3.2 Functionality of the “Buildsec” Library

The core component of the system architecture shown in Figure 3.1 is the buildsec library which will be further described in this section. Buildsec library has two main components namely; “Annotation Library” and “Annotation Processor”. Both these components are jar files.

As mentioned previously, “buildsec” annotation is declared inside the “Annotation Library” and used in the mobile application source code where the vulnerabilities need to be verified. During the compile time, when the compiler invokes the “Annotation Processor” the “buildsec” annotation helps the “Annotation Processor” to retrieve the source code in class level.

“Annotation Processor” consists of the five components namely; “Source code Reader, “Manifest Parser”, “Configurator”, “Verification Module” and “Report Generator”. As illustrated in Figure 3.3 these components interact with each other to verify and fix source code vulnerabilities in the mobile application source code.

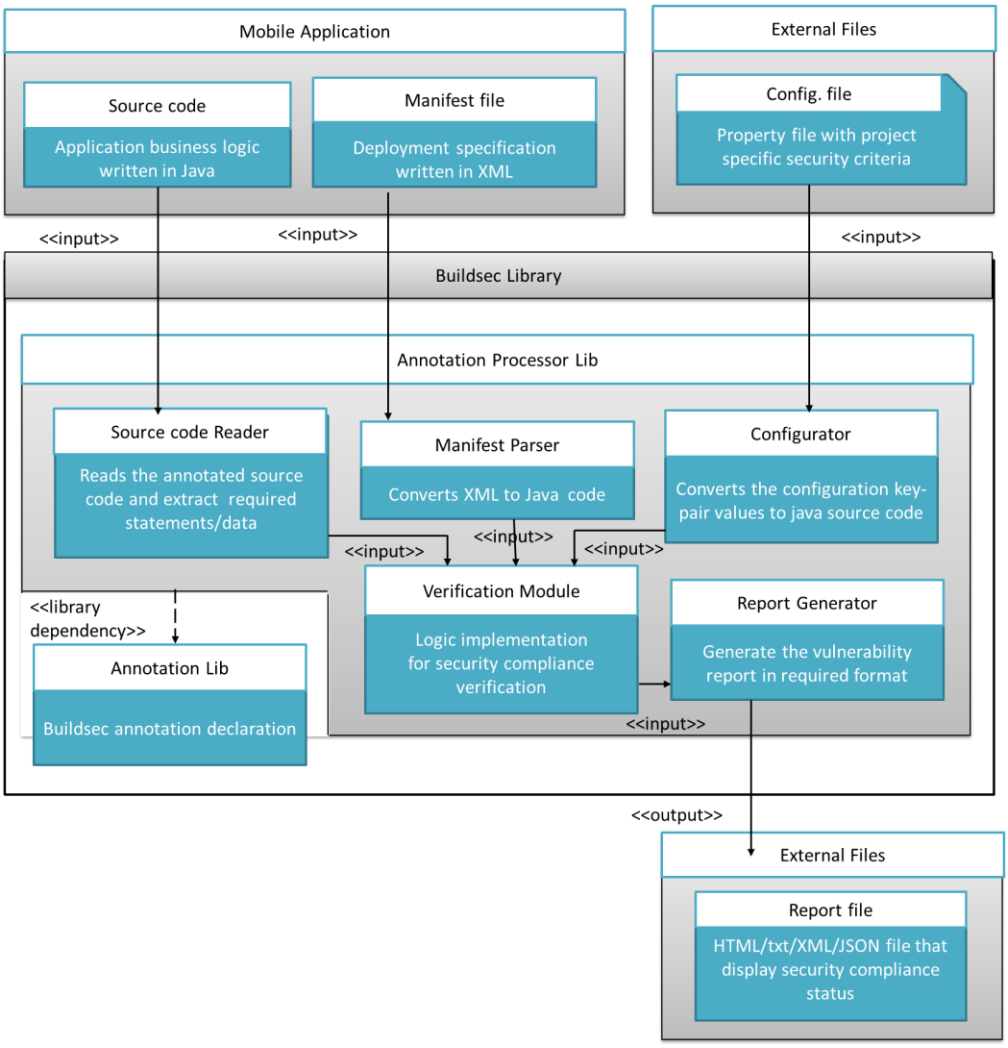

Figure 3.3: Functionality of the Buildsec library

Source Code Reader

Source code reader reads the mobile application source code annotated with @buildsec annotation and extracts the statements required for security verifications.

Manifest Parser

Manifest parser retrieves the whole content of the android manifest file and converts it to a java code.

Configurator

Configurator reads the configuration file which is in key-value pairs and converts it to a java code.

Verification Module

Verification module is a set of java classes which contain the verification logics for security vulnerabilities. This module retrieves the annotated source code from the source code reader, android manifest converted to a java code from manifest parser, configuration details converted to a java code from the configurator and input them to verification logic functions to verify and fix source code vulnerabilities.

Report Generator

Report generator retrieves the details of the vulnerabilities from the verification module and output the compliance status report in HTML/XML/text/JSON format.

3.3 Deployment

The minimum system requirements required to utilize the “buildsec” library in the android platform are listed below.

- Android Studio 3.0
- Android SDK
- JDK 1.7

To import “buildsec” library into “Android Studio” and build mobile application, the following steps can be followed.

1. Open your project in Android Studio.
2. Download the buildSec.jar and buildSecProcessor.jar, from the following github location using git or as a zip archive and unzip it.
   https://github.com/iamarasekera/build-secure
3. Copy the jar files in to the lib folder in the *‘app’* module of your android project.
4. Go to File -> Import Module and import the library as a module.
5. Right click your app in project view and select "Open Module Settings".
6. Click the "Dependencies" tab and then the '+' button.
7. Select "Module Dependency".
8. Select "buildSec.jar Library".
9. Select “buildSecProcessor.jar”.
10. Edit your project's “build.gradle” file to add the following lines in the "defaultConfig" section:

```
javaCompileOptions{
    annotationProcessorOptions{
        includeCompileClasspath true
    }
}
```

```
android {
    signingConfigs {
    }
    compileSdkVersion 26
    defaultConfig {
        applicationId "com.example.ishara.buildsec_demoapp"
        minSdkVersion 19
        targetSdkVersion 26
        versionCode 1
        versionName "1.0"
        javaCompileOptions {
            annotationProcessorOptions {
                includeCompileClasspath true
            }
        }
    }
```

Figure 3.4: Sample build.gradle file

Note: The other settings given in “build.gradle” file can differ from what is shown in Figure 3.4.

11. Clean and build the android project.

CHAPTER 4
TESTING AND EVALUATION

4.1 Test Approach

The following approach was followed in order to evaluate the implemented system against manual testing.

1. Derive test cases

   A set of test cases are derived out of few use cases that can be used to test the accuracy of the existing security verification modules.

2. Set security compliance level

   As “buildsec” currently supports a limited number of verification methods, only two levels of security compliance are set. If any of the defined security criteria are not met, the entire application will be marked as security noncompliance.

3. Define test case pass rate

   The expected result of each test case is required to be aligned with the result expected in manual testing (i.e. code review) when the same test cases are executed.

4.2 Test Report

The test cases executed and the test result obtained in each case is given below. The "buildsec" exhibited 100% pass rate for all the test scenarios.

Test Case Id	1
Use case	Unauthorized request to access uses permission
Description	Uses permission is declared in the manifest file but no relevant code in the java source code
Sample Code	AndroidManifest.xml <?xml version="1.0" encoding="utf-8"?> <manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.example.ishara.buildsec_demoapp"> **<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />** <application android:allowBackup="false" android:icon="@mipmap/ic_launcher" android:label="@string/app_name" android:roundIcon="@mipmap/ic_launcher_round" android:supportsRtl="true" android:theme="@style/AppTheme"> <activity android:name=".MainActivity" android:label="@string/app_name" android:theme="@style/AppTheme.NoActionBar"> <intent-filter> <action android:name="android.intent.action.MAIN" /> <category android:name="android.intent.category.LAUNCHER" /> </intent-filter> </activity> </application> </manifest>
Expected Result	Security status : non-compliance
Actual Result	Security status : non-compliance Remove unnecessary ACCESS_FINE_LOCATION permission from the manifest file and recompile the source code
Test Status	Pass

Test Case Id	2
Use case	Missing request to access uses permission
Description	The relevant uses permission is not declared in the manifest file but the source code contains methods that requires the missing permission

Sample Code	AndroidManifest.xml
	
	<?xml version="1.0" encoding="utf-8"?>
	<manifest xmlns:android="http://schemas.android.com/apk/res/android"
	  package="com.example.ishara.buildsec_demoapp">
	  <application
	  android:allowBackup="false"
	  android:icon="@mipmap/ic_launcher"
	  android:label="@string/app_name"
	  android:roundIcon="@mipmap/ic_launcher_round"
	  android:supportsRtl="true"
	  android:theme="@style/AppTheme">
	  <activity
	  android:name=".MainActivity"
	  android:label="@string/app_name"
	  android:theme="@style/AppTheme.NoActionBar">
	  <intent-filter>
	  <action android:name="android.intent.action.MAIN" />
	  <category android:name="android.intent.category.LAUNCHER" />
	  </intent-filter>
	  </activity>
	  </application>
	</manifest>

	MainActivity.java
	
	public static Camera **getCameraInstance()**{
	  Camera c = null;
	  try {
	  c = **Camera.open()**; // attempt to get a Camera instance
	  }
	  catch (Exception e){
	  // Camera is not available (in use or does not exist)
	  }
	  return c; // returns null if camera is unavailable
	}

Expected Result	Security status : non-compliance
Actual Result	Security status : non-compliance Inject the required CAMERA permission to the manifest file and recompile the source code
Test Status	Pass

Test Case Id	3
Use case	Proper access to uses permission
Description	As per the source code, the relevant uses permission declared in the manifest file
Sample Code	AndroidManifest.xml <?xml version="1.0" encoding="utf-8"?> <manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.example.ishara.buildsec_demoapp"> **<uses-permission android:name="android.permission.CAMERA" />** <application android:allowBackup="false" android:icon="@mipmap/ic_launcher" android:label="@string/app_name" android:roundIcon="@mipmap/ic_launcher_round" android:supportsRtl="true" android:theme="@style/AppTheme"> <activity android:name=".MainActivity" android:label="@string/app_name" android:theme="@style/AppTheme.NoActionBar"> <intent-filter> <action android:name="android.intent.action.MAIN" /> <category android:name="android.intent.category.LAUNCHER" /> </intent-filter> </activity> </application> </manifest>

	MainActivity.java
	
	public static Camera **getCameraInstance()**{
	Camera c = null;
	try {
	c = **Camera.open()**; // attempt to get a Camera instance
	}
	catch (Exception e){
	// Camera is not available (in use or does not exist)
	}
	return c; // returns null if camera is unavailable
	}

Expected Result	Security status : compliance
Actual Result	Security status : compliance
Test Status	Pass

Test Case Id	4
Use case	Enable Auto-Backup in the application
Description	The auto backup is enabled in the application but the security compliance criteria is to disable it.
Sample Code	AndroidManifest.xml <?xml version="1.0" encoding="utf-8"?> <manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.example.ishara.buildsec_demoapp"> <uses-permission android:name="android.permission.CAMERA" /> <application **android:allowBackup="true"** android:icon="@mipmap/ic_launcher" android:label="@string/app_name" android:roundIcon="@mipmap/ic_launcher_round" android:supportsRtl="true" android:theme="@style/AppTheme"> <activity android:name=".MainActivity" android:label="@string/app_name" android:theme="@style/AppTheme.NoActionBar"> <intent-filter> <action android:name="android.intent.action.MAIN" /> <category android:name="android.intent.category.LAUNCHER" /> </intent-filter> </activity> </application> </manifest>
Expected Result	Security status : non-compliance
Actual Result	Security status : non-compliance Set the value of the "allowBackup" tag to false
Test Status	Pass

Test Case Id	5
Use case	Disable Auto-Backup in the application
Description	The auto backup is disabled in the application as per the security compliance criteria.
Sample Code	AndroidManifest.xml <?xml version="1.0" encoding="utf-8"?> <manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.example.ishara.buildsec_demoapp"> <uses-permission android:name="android.permission.CAMERA" /> <application **android:allowBackup="false"** android:icon="@mipmap/ic_launcher" android:label="@string/app_name" android:roundIcon="@mipmap/ic_launcher_round" android:supportsRtl="true" android:theme="@style/AppTheme"> <activity android:name=".MainActivity" android:label="@string/app_name" android:theme="@style/AppTheme.NoActionBar"> <intent-filter> <action android:name="android.intent.action.MAIN" /> <category android:name="android.intent.category.LAUNCHER" /> </intent-filter> </activity> </application> </manifest>
Expected Result	Security status : compliance
Actual Result	Security status : compliance
Test Status	Pass

Test Case Id	6
Use case	Keep debugging logs entries in the source code
Description	Log entries used for debugging and error handling purpose are left in the source code

Sample Code	MainActivity.Java
	
	@Override
	public void onBackPressed() {
	DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
	if (drawer.isDrawerOpen(GravityCompat.START)) {
	drawer.closeDrawer(GravityCompat.START);
	Log.d("onBackPressed", "if block for closeDrawer is executed");
	} else {
	super.onBackPressed();
	Log.d("onBackPressed", "else block is executed");
	}
	}

Expected Result	Security status : non-compliance
Actual Result	Security status : non-compliance Remove all log entries and recompile
Test Status	Pass

Test Case Id	7
Use case	No debugging logs entries are in the source code
Description	All log entries used for debugging and error handling purpose are removed in the source code

Sample Code	MainActivity.Java
	
	@Override
	public void onBackPressed() {
	DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
	if (drawer.isDrawerOpen(GravityCompat.START)) {
	drawer.closeDrawer(GravityCompat.START);
	} else {
	super.onBackPressed();
	}
	}

Expected Result	Security status : compliance
Actual Result	Security status : compliance
Test Status	Pass

CHAPTER 5
CONCLUSION

5.1 Summary

This research project had two main objectives to be accomplished via reach component and implementation. A comprehensive research study has been conducted under the research component to identify the security risks, how they are injected to the mobile source code and what actions can be taken to eliminate them. An evaluation of the existing testing approaches, tools and instruments was done to understand to which extend these tools can be helpful to fix source code vulnerabilities. Similar research conducted towards designing frameworks, architecture and infrastructure for secure mobile application development were also studied to understand the research gaps. The aforementioned research findings were helpful in designing the architecture of "buildsec" library. Learnings on SSC were helpful to understand how SSCs are used in SDLC model to overcome the security threats and risks. The causes and elimination methods of source code vulnerabilities were identified as another research outcome and applied that knowledge to implement the modules of "buildsec" library.

5.2 Contribution

As per the research findings, it is clear that the developer mistakes and insecure development practices cause source code vulnerabilities and there is no proven mechanism to halt it. Most of the security violations are discovered after the application is released for testing. Hence, there are excessive iterations running between the development and testing phase increasing the cost of the project due to the increased effort, time and re-work. The "buildsec" library, which is the outcome of this research project, enforces secure mobile application development. It ensures that a security non-compliance build is not released to the testing team and thereby reduces the project cost caused due to repeated development and testing efforts. "buildsec" is available as an open source project in the github location, https://github.com/iamarasekera/build-secure and can be integrated to the mobile platform without much effort.

5.3 Limitations and Future Work

The implementation of the “buildsec” library is currently limited to android platform. Android platform is chosen for development as it is open source and is most feasible for the implementation. However the same approach or the architecture can be implemented in iOS and web based mobile platforms as well as a future development.

“Buildsec” currently supports a limited number of security verification methods. As mentioned in Section 3.1 this library can easily be extended to add new modules as well as to enhance the existing modules.

There is also a possibility to support offline compilation using the same library to verify already compiled source code if the “Annotation Processor” is invoked by a testing module (e.g., automated testing) instead of the compiler. This can also be suggested as a future development for the “buildsec” library.

REFERENCES

[1] R. Van Der Meulen and J. Rivera, "Gartner Says Worldwide Traditional PC, Tablet, Ultramobile and Mobile Phone Shipments On Pace to Grow 7.6 Percent in 2014," *Gartner.com*, 2014. [Online]. Available: https://www.gartner.com/newsroom/id/2645115. [Accessed: 19-Nov- 2017].

[2] Sophos, "Security Threat Report 2013: New Platforms and Changing Threats," *Sophos*, 2013. [Online]. Available: https://www.sophos.com/en-us/medialibrary/PDFs/other/sophossecuritythreatreport2013.pdf. [Accessed: 19-Nov-2017].

[3] F-Secure, "Mobile Threat Report 2013," *F-Secure*, 2013. [Online]. Available: https://www.fsecure.com/static/doc/labs_global/Research/Mobile_Threat_Report_Q3_2013.pdf. [Accessed: 19-Nov- 2017].

[4] Symantec, "Internet Security Thread Report 2014," *Symantec*, 2014. [Online]. Available: https://www.symantec.com/security_response/publications/threatreport.jsp. [Accessed: 19-Nov- 2017].

[5] Y. Lin, C. Huang, M. Wright, and G. Kambourakis, "Mobile Application Security," *Computer*, vol. 47, no. 6, pp. 21–23, 2014.

[6] M. Daud, "Secure Software Development Model: A Guide for Secure Software Life Cycle," in *The International MultiConference of Engineers and Computer Scientists*, Hong Kong, March 2010, pp. 1-5.

[7] D. Gilliam, T. Wolfe, J. Sherif, and M. Bishop, "Software Security Checklist for the Software Life Cycle," In *Proc. 12th IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprise*, June 2003, pp. 243–248.

[8] D. Gilliam and J. Powell, "Integrating a Flexible Modeling Framework (FMF) with the Network Security Assessment Instrument to Reduce Software Security Risk," in *Proc. 11th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, June 2002, pp. 153-160.

[9] G. McGraw, "Software Risk Management for Security," *IEEE Computer*, 32(4), April, pp. 103-105, 1999.

[10] A. Jaquith, "The Security of Applications: Not All Are Created Equal," Research Report, *@Stake.Inc*., February 2002. [Online]. Available: http://www.atstake.com/research/reports/acrobat/atstake_app_unequal.pdf. [Accessed: 19-Nov-2017].

[11] K. S. Hoo, A. W. Saudbury and A. Jaquith, "Tangible ROI through Secure Software Engineering," *Secure Business Quarterly*, Q4, vol. 1, no. 2, 2001.

[12] Open Web Application Security Project, "OWASP Mobile Security Project Top 10 Mobile Risks," *owasp.org*, 2014. [Online]. Available: https://www.owasp.org/index.php/OWASP_Mobile_Security_Project#tab=Top_Ten_Mobile_Risks. [Accessed: 19-Nov- 2017].

[13] C. Wysopal, "Mobile App Top 10 List," *Veracode*, 2010. [Online]. Available: http://www.veracode.com/blog/2010/12/mobile-app-top-10-list. [Accessed: 19-Nov-2017].

[14] J. Shin, D. Kim, K. Han and H. Kim, “A Study on the Security Checklist Improvements to improve the Security in the Mobile Applications Development,” *Journal of Digital Convergence*, vol. 12, issue 8, pp.113-127, 2014.

[15] United States Department of Homeland Security, *Study on Mobile Device Security*. USA Department of Homeland Security, 2017.

[16] WhiteHat Security, “*Integrating Application Security into the Mobile Software Development Lifecycle,”* WhiteHat Security, Santa Clara, 2015.

[17] B. N. Harsha, “Mobile Application Security Testing - Launch Secure Applications,” *idexcel*, 2017. [Online]. Available: http://www.idexcel.com/resources/whitepapers. [Accessed: 19-Nov- 2017].

[18] A. K. Jain, D. Shanbhag, "Addressing security and privacy risks in mobile Applications", *IT Professional*, vol. 14, no. 5, pp. 28-33, 2012.

[19] J. Burns, “Developing secure mobile applications for android - an introduction to making secure android applications,” *iSEC*, Oct. 2008. [Online]. Available: https://www.nccgroup.trust/globalassets/our-research/us/whitepapers/isec_securing_android_apps.pdf. [Accessed: 19-Nov- 2017].

[20] Android Developers, “App Security Best Practices,” *developer.android.com*, 2017. [Online]. Available: https://developer.android.com/topic/security/best-practices.html. [Accessed: 19- Nov- 2017].

[21] Android Developers “*Security Tips”*, *developer.android.com*, 2017. [Online]. Available: https://developer.android.com/training/articles/security-tips.html. [Accessed: 19- Nov- 2017].

[22] Apple Developer, “Introduction to Secure Coding Guide,” *developer.apple.com*, 2017. [Online]. Available: https://developer.apple.com/library/mac/documentation/Security/Conceptual/SecureCodingGuide/Introduction.html. [Accessed: 19- Nov- 2017].

[23] Center for Democracy & Technology, “Best Practices for Mobile Applications Developers,” *cdt.org*, 2011. [Online]. Available: https://cdt.org/blog/best-practices-for-mobile-applications-developers/. [Accessed: 19-Nov- 2017].

[24] J. Sun, D. Howie, A. Koivisto and J. Sauvola, “A Hierarchical Framework Model of Mobile Security,” In *Proc.12th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications,* Cat. No.01TH8598, vol. 1, 2011.

[25] G. Florian and Furtmüller, “An Approach to Secure Mobile Enterprise Architectures,” *International Journal of Computer Science*, vol. 10, no. 1, 2013.

[26] S. Chakraborti, D. Acharjya and S. Sanyal, “Application Security framework for Mobile App Development in Enterprise setup,” *International Journal of Advanced Networking and Applications*, vol.1, 2015.