BUILD-SECURE: A SOFTWARE PROGRAM TO ENSURE SECURE MOBILE APPLICATION DEVELOPMENT

P. A. I. U. Amarasekera

148202X

M.Sc. in Computer Science

Department of Computer Science and Engineering

University of Moratuwa

Sri Lanka

December 2017

BUILD-SECURE: A SOFTWARE PROGRAM TO ENSURE SECURE MOBILE APPLICATION DEVELOPMENT

P. A. I. U. Amarasekera

148202X

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

Department of Computer Science and Engineering

University of Moratuwa

Sri Lanka

December 2017

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. Reserch Project.

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

Dr. Malaka Walpola Date

ABSTRACT

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

Hence the intention of this project 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 the mobile application is submitted for testing. This will reduce the testing effort as well as development re-work to fix the security issues found out in the testing phase.

ACKNOWLEDGEMENTS

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

My deepest gratitude and admiration goes to my adviser, 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 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

DHS	Department of Homeland Security
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hyper Text Transfer Protocol with Secure Sockets Layer
IPA	IPhone Application Archive
IPC	Inter Process Communication
MAM	Mobile Application Management
MDM	Mobile Device Management
NIST	National Institute of Standards and Technology
OS	Operating System
OWASP	Open Web Application Security Project
PIN	Personal Identification Number
QA	Quality Assurance
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
TLS	Transport Layer Security
UAT	User Acceptance Testing
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 shifted their targets to mobile computing platforms [1]. As concluded by Sophos [2] in 2013 Android is the biggest target for security attacks, and as reported by F-secure [3], the number of mobile malware samples grew from several hundreds to more than 50,000 in just two years. The ubiquitous and popular use of mobile devices has made mobile application security a persistent issue. Since, mobile devices contain large amounts of sensitive personal information; they have become attractive targets for attackers seeking financial gain. According to Symantec [4] 57 percent of adult mobile users are still unaware that security solutions exist for mobile devices. Reports also show that about half of the security attacks are triggered to track users and to steal personal information. The limited computational power and the restricted user interface have made it easier for attackers to hide their malicious activities.

A security environment in mobile application is composed of 3 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 app 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 lot 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 percent 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 and vaccine management services such as MDM and 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. Several recent studies have shown that the risk of not integrating security into the software life cycle can have highly negative impacts on a company. Both Citigal and @Stake have done studies that show that integrating security in the software life cycle has proven benefits both in cost and reputation. [9, 10, 11]

In order to identify security vulnerabilities during the testing phase, different tools and mechanisms are available. Most of these tools and instruments which are used in the life cycle are from other formal disciplines such as reliability and safety and include modeling, code-auditing, fault/attack trees and fault injection, property-based testing, boundary testing, etc. [7]. In high-level, those testing tools can be categorized into 3 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 security analysts to understand the behavior of running systems so that they can identify potential issues.[17] Proxies that allow security analysts to observe and potentially change the communications between mobile application clients and supporting Web services are the most common types. [17]

3. Forensic testing tools that allow application security analysts to examine the artifacts left behind by the application once it has been run. [17] These source code scanners/analyzers check for potential issues in the source code that would cause security risks.

As fixing applications post-attack is expensive in both financially and company reputation, integrating security into the life cycle is proven to be a revenue loss preventative [8]. Hence, it is highly cost effective for an organization to document their security policies, requirements, guidelines and procedures. Additionally, it is also essential to educate and train Software Engineers, developers, System Administrators etc.

The mobile software development life cycle consists of the following steps, which are performed as a part of a repeatable process called the phases of software development:

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

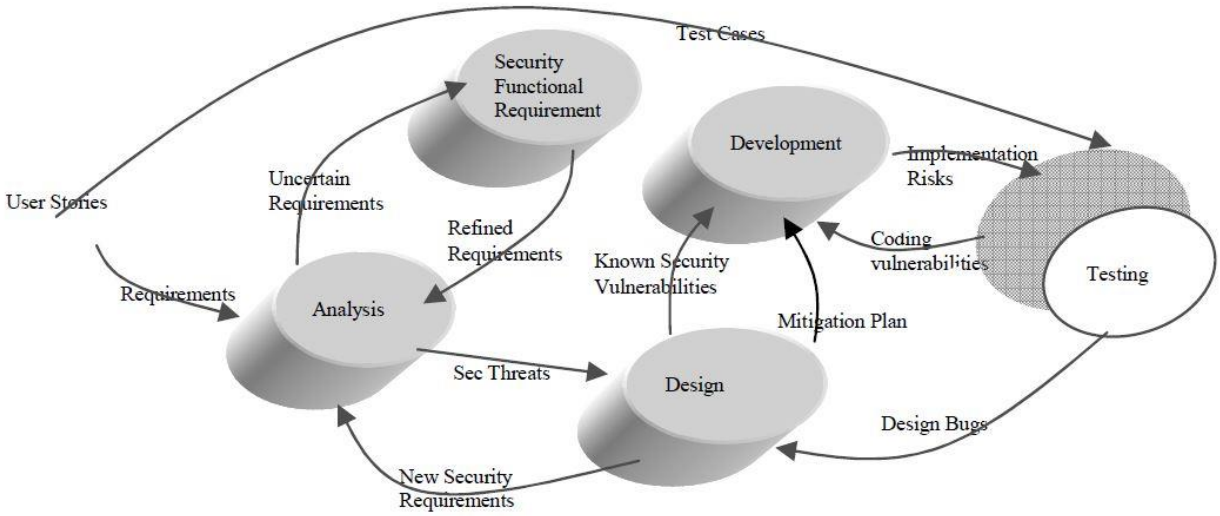
Security testing has been typically performed during the UAT or QA phase near the end of development cycle. As a result, testing in only one phase of the SDLC provides a limited view of security. Unfortunately, a development team following this paradigm would likely not have identified security defects in the code until late in the development life cycle and after the code was compiled into a functional application. Finding security defects at this late stage of development is highly inefficient and very 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 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 given for testing. This will reduce the testing effort as well as development re-work to fix the security issues found out in the testing phase.

1.2 Secure Software Life Cycle (SSLC) Model

Secure Software Development Model [6] is a research outcome where an iterative lifecycle for secure software development is introduced to mitigate these 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 X shown below illustrates the iterative life cycle for secure software development [6].



During the requirement phase, security engineers elicit and derive security requirements by different methods, e.g. user stories, abuse cases, functional and non-functional security requirements, and etc. In analysis phase these security requirements are refined by a security functional requirements (SFR) module. Then the design phase shapes all the requirements into reality by developing a threat model to identify threats, vulnerabilities and their countermeasures. The vulnerability analysis and identifying entry and exit points of all possible threats that an attacker can exploit from these points are the main activities of this phase. As per the research outcome, below are the common vulnerability areas that need to be addressed [6].

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

Vulnerability areas shown in figure XX can be taken as security use cases and their countermeasures are then figured out. Once we have identified all the attacks and vulnerabilities now system is ready for implementation phase. 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 at the development phase. Hence, this model operates in a way where the identified vulnerabilities are addressed by developers and send it for testing where there is no guarantee that all vulnerabilities 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 is considered vital though as it plays an important role in identifying security flaws before the application release. 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].

1.3 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 an instrumental guide that helps the software development teams to integrate security into the software life cycle was presented. Due to its criticality, security should be integrated as a formal

approach in the software life cycle and as a result, both a software security checklist and assessment tools are incorporated into this life cycle process.[7]

The first step towards implementing a SSC is to perform a security risk analysis and then identify security risks and requirements. The second step is to use an SSC instrument for all phases of the life cycle. The risk analysis and requirements should then drive the rest of the life cycle with traceable and verifiable security requirements throughout. The last step of the SSC addresses the tools to assess security during the life cycle process. [7] There is a separate section that discusses these tools and instruments.

SSC covers the critical areas of requirements gathering and specification, design and code issues, and maintenance and decommissioning of software and systems. It has two main phases as given below. [7]

- Phase 1 - a security checklist for the software development and maintenance life cycle
- Phase 2 – a security checklist for the external release of software.

Both these checklists are important to the developer as developer is responsible for not only software development and maintenance but also for creating the products to be released. (Products include software, documentation, test data, configuration files, etc.).

Table 1 below describes some critical areas that can be used for generating an SSC for software development life cycle. It is provided as one example for generating an SSC hence; the list can be extended, modified, or even replaced by the security risks or the vulnerabilities identified in the requirement analysis or the design phase. [7] The mobile security risks and vulnerabilities in mobile applications are further discussed in section XX.XX.

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.

< Table 1: Items for Potential Consideration and Inclusion in a Software Security Checklist (SSC) >

Table 2 below provides an example of an Initial Start of a Security Checklist for the software that is developed for release external to the organizational environment.

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	. . .

<Table 2: Example of an Initial Start of a Security Checklist for the External Release of Software>

Below is another set of security vulnerability diagnostic items presented by a joint effort of Computer Science Departments of Konkuk and Shamyook Universities based on 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

All the aforementioned checklist items are not the sole responsibilities 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 XXX section the developer responsibilities to handle security vulnerabilities in the source code is further discussed.

1.4 Mobile Application Security Risks

Moving to "Mobile Application Security", which is the core topic of this research it is necessary to understand what are the security risks associated with mobile application development. There are several examples of vulnerabilities in mobile applications that expose the users to excessive risk as identified by different authorities and organizations that has performed continuous research and conducted studies on this subject. 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 OWASP Mobile Security Project is a centralized resource intended to give developers and security teams the resources they need to build and maintain secure mobile applications. The Top 10 Mobile Application Risks, or "Mobile App Top 10" for short, is designed to educate developers and security professionals about the mobile application behavior that puts users at risk. [12]

1. Insecure data storage

   This risk occurs when sensitive data is stored on a device or when cloud-synced data is left unprotected. It's generally the result of not encrypting sensitive data, caching information not intended for long-term storage, allowing global file permissions, or failing to leverage best practices for a particular platform. This in turn leads to the exposure of sensitive information, privacy violations, and noncompliance.

2. Weak server side controls

Failure to implement proper security controls such as patches and updates or secure configurations, changing default accounts, or disabling unnecessary backend services can compromise data confidentiality and integrity.

3. Insufficient transport layer protection

Mobile applications often use the HTTP protocol for client-server communication, which communicates all information in plain text. Even when they provide transport-layer security through use of the HTTPS protocol, if they ignore certificate validation errors or revert to plain-text communication after a failure, they can jeopardize security by revealing data or facilitating data tampering through man-in-the middle attacks.

4. Client side injection

Apart from the known injection attacks such as XSS, HTML injection, and SQL injection applicable to mobile Web and hybrid apps, mobile apps are witnessing newer attacks such as abusing the phone dialer, SMS, and in-app payments.

5. Poor authorization and authentication

Poor authorization and authentication schemes relying on device identifiers for security—such as the International Mobile Equipment Identifier, International Mobile Subscriber Identity, or Universally Unique ID (UUID) values are a recipe for failure. They can lead to broken authentication and privilege-access issues.

6. Improper session handling

Sessions that have long expiration times or that use device identifiers as the session ID pose security risks, such as privilege escalation and unauthorized access.

7. Security decisions via untrusted inputs

If applications make security decisions via user input, then the used inputs can be leveraged by malware or client-side injection attacks for various nefarious purposes, such as consuming paid resources, exfiltration data, or escalating privileges. For example, attackers have abused URL schemes in the iOS and intents in Androids.

8. Side channel data leakage

Programmatic flaws or the failure to disable insecure OS features in applications can result in sensitive data ending up in Web caches, global OS logs, screenshots (an iOS back-grounding issue), and temp directories. The data is then up for grabs for malware or an attacker who steals your phone.

9. Broken cryptography

This risk emanates from insecure development practices, such as using custom instead of standard cryptographic algorithms, assuming that encoding and obfuscation are equivalent to encryption, or hardcoding cryptographic keys into the application code itself. Such practices can result in a loss of data confidentiality or privilege escalation.

10. Sensitive information disclosure

Hardcoding sensitive information such as login credentials, shared secret keys, access tokens, and sensitive business logic into the application code means that an attacker might be able to access such information through reverse engineering, which is fairly trivial. Once the attacker has this information, it's easy to access more sensitive data.

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

Veracode lists down mobile application security risks into 2 main categories. The category of “Malicious Functionality” is a list of unwanted and dangerous behaviors of mobile applications that put user at risk. The category of “Code Vulnerabilities” is errors in design or implementation that expose the mobile device data to interception and retrieval by attackers. [13]

A. Malicious Functionality

1. Activity monitoring and data retrieval

The possibility of attackers to monitor and intercept information and data in real time as it is being generated on the device.

Example:

- Sending each email sent on the device to a hidden 3rd party address
- Listening in on phone calls or simply open microphone recording
- Stored data, contact list or saved email messages retrieved

2. Unauthorized dialing, SMS and payments

The ability the attackers gain to directly monetize a compromised device.

Examples:

- Premium rate phone calls - By including premium dialing functionality into a Trojan app the attacker can run up the victim’s phone bill and get the mobile carriers to collect and distribute the money to them.
- Mobile payments - Mobile devices can also be used to purchase items, real and virtual, and have the cost billed on the customer’s mobile bill.
- SMS text message as a vector for worms - unauthorized SMS text message is as a spreading vector for worms. Once a device is infected a worm can send SMS text messages to all contacts in the address book with a link to trick the recipient into downloading and install the worm

3. Unauthorized network connectivity (exfiltration or command & control)

As mobile devices are designed for communication there are many potential vectors that a malicious app can use to send data to the attacker. A full function malicious program will often allow the attacker to direct commands to the spyware to for instance turn on the microphone or grab a data file at a particular time. Email, SMS, HTTP GET/POST,TCP socket, UDP socket, DNS exfiltration and Bluetooth are some of the such communication channels that attackers can use for exfiltration and command and control.

4. UI Impersonation

This is similar to phishing attacks where users are tricked to click on a link in a browser or enter their credentials. This can take the form of a web view application which presents a native mobile UI as a proxy to a native web app or a malicious application popping up UI that impersonates that of the phone's native UI of a legitimate application.

5. System modification (rootkit, APN proxy config)

When there is a malicious applications present in the device, they will often attempt to modify the system configuration to hide their presence. E.g. modifying the device proxy configuration and modifying the Access Point Name (APN). This is often called rootkit behavior. These configuration changes will make certain other attacks possible.

6. Logic or time bomb

This is a classic backdoor techniques that trigger malicious activity based on a specific event, device usage or time such as certain hours of the day or days of the week, upon receipt of an email or SMS from a particular sender or when a phone call is made.

B. Code Vulnerabilities

1. Sensitive data leakage

Sensitive data leakage can be either inadvertent or side channel. Poorly implemented source code can expose the sensitive data such as location, owner's information: name, number and device ID, authentication credentials, authentication tokens etc. to third parties.

2. Unsafe sensitive data storage

Mobile apps often store sensitive data. E.g. Banking and payment system PIN numbers, credit card numbers, or online service passwords etc. These sensitive data should always be stored encrypted. A strong cryptography should be used to prevent data being stored in a manner that it allows unauthorized retrieval. Storing sensitive data on removable media such as a micro SD card without encryption is especially risky.

3. Unsafe sensitive data transmission

It is important to ensure that sensitive data is encrypted during transmission. Mobile devices are especially susceptible as they use wireless communications exclusively for data transmission and often public WiFi. SSL is proven to be one of the best ways to secure sensitive data in transit.

4. Hardcoded passwords/keys

Hardcoded passwords or keys are used by developers as a shortcut to make the applications easier to implement, support, or debug. However, if the hardcoded passwords are discovered through reverse engineering or other means, everybody will have access to it (e.g. backdoor passwords for router maintenance). This will make the security of the application to fail and the system(s) being authenticated to may suffer due to trust assumptions.

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 as listed below.

1. Third party applications running in jailbroken or rooted devices

   Users may attempt to root or jailbreak their device to gain access to application stores that might otherwise be inaccessible. The root or jailbreak process often places the device in a degraded security state that could be taken advantage of by attackers by allowing the 3rd parties to gain access to the device and run malicious application or perform unauthorized activities.

2. Insecure network communication

   If network traffic between an application and a remote server is not securely encrypted an attacker positioned on the network can eavesdrop on the connection, including obtaining sensitive data such as login credentials. Attacker will also get the opportunity to alter data as it traverses the path, resulting in delivery of compromised information. The failure to properly authenticate the identity of the remote server when connecting also creates the opportunity for man-in-the-middle attacks.

3. Sharing of data between trusted apps

   Apps may share data with external resources such as Dropbox without the user's awareness. This is clearly violation of data privacy and user will be at risk of losing data and allowing a 3rd party to gain access to his personal data such as images, password etc.

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

Applications with this vulnerability can lead to exposure of sensitive information, often without the user's knowledge. This is referred to storing personal data (contacts, profile, message logs) in an unencrypted format and with improperly assigned insecure file permissions that allowed anyone or any app to read them.

5. Sensitive information written to system and console log

Applications for Android and iOS have been found that write sensitive information into plain text log files are read by 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

Web browser applications can be exploited by attackers as an entry point to gain access to a mobile device. Newer versions of Android and iOS include security architecture improvements designed to make it more difficult for an attacker to exploit web browser vulnerabilities. Backing up data files to an external location and performing a factory reset of the phone can mitigate the attack, but at an expense of time and effort. Updating to a more robust browser is also recommended activity to prevent this.

7. Vulnerabilities in third-party libraries

Third-party software is reusable components that may be distributed freely or offered for a fee to other software vendors. Software development by component or modules is often considered more efficient, and third-party libraries are routinely used across the industry. However, when a library is flawed it can introduce vulnerabilities in any app that includes or makes use of that library. Depending on the pervasiveness of the library, its use can potentially affect thousands of apps and millions of users.

8. Cryptographic Vulnerabilities

Cryptographic vulnerabilities can occur via failure to use cryptographic protections for sensitive data, by the improper implementation of a secure cryptographic algorithm, or the use of a proprietary cryptographic technique that can be more easily cracked than those validated and recommended for use by NIST. The net result to users, however, is likely to be the same, sensitive information that is presumed secure is potentially exposed to unauthorized users.

1.5 Security Assessment Tools and Instruments Used in SDLC

Security assessment tools are valuable and useful in producing secure software. A number of tools are currently available for use in the software life cycle that can be used to assess and test system and software security beginning from the requirements phase through to the operation of the software. Testing is typically performed during the UAT or QA phase of the life cycle. 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 used to test either the source code or the application binary. These are good for identifying certain types of vulnerabilities in how the code will run on the device, usually associated with dataflow and buffer handling. Some commercial static security analysis tools and services have the capability to test mobile application code.

2. Dynamic testing tools

Dynamic testing tools allow security analysts to observe the behavior of running systems in order to identify potential issues. The most common dynamic analysis tools used in mobile app security testing are proxies that allow security analysts to

observe and potentially change communications between mobile application clients and supporting Web services. With proxy tools, security analysts can reverse engineer communication protocols and craft potentially malicious messages that would never be sent by legitimate mobile clients. This allows the messages to attack the server-side resources that are a critical component of any nontrivial mobile application system.

3. Forensic testing tools

Forensic tools allow security analysts to examine artifacts that are left behind by an application after it has been run. Common things analysts might look for include hard-coded passwords or other credentials stored in configuration files, sensitive data stored in application databases and unexpected data stored in Web browser component caches. Analysts can also use forensic tools to look at how components of mobile applications are stored on the device to determine if available operating system access control facilities have been properly used.

Security testing in mobile applications needs to examine a number of different aspects as mobile applications can have complicated threat models. 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 covers static, dynamic and forensic testing to ensure that mobile applications are tested in every security aspect. Below is a set of freely available, open source and inbuilt security assessments tools available for mobile platforms.

Tool/Instrument	Type	Supported Mobile Platform	Usage
Clang Static Analyzer	Static	iOS	This a static analysis tool for C, C++ and Objective-C programs. This can be used the Objective-C support to test for certain quality and security errors in iOS-based applications, and they can be run in both from the command line and from 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 - This can be used to generates DEX assembly code from an Android DEX application binary dex2jar - This converts DEX application binaries to standard Java JAR files. FindBugs - This 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 bytecode back into Java source code which helps to review or scan the code for vulnerabilities
OWASP Zed Attack Proxy.	Dynamic	Android and iOS	The framework 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	iPad File Explorer allows to browse files structure on iOS device. Using this, both Media files and App data can be listed out in different views. This is also able to acess device storage file system of jail broke iOS 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	A virtual machine that contains a number of open source tools specific to mobile application security testing, forensics and data recovery, and malware analysis.
Mobile Security Framework (MobSF)	Static and Dynamic	Android and iOS	An automated penetration testing framework for Android and iOS apps including static and dynamic analysis and web API testing capabilities.
Mitmproxy	Dynamic	Android and iOS	This is an interactive man-in-the-middle proxy for HTTP and HTTPS with a console interface which allows a user to intercept and modify requests and responses exchanged between an app and backend services in order to inspect any data transferred.
Drozer	Dynamic	Android	Identifies security vulnerabilities in Android apps and devices and supports the use and sharing of public exploits. It discovers and interact with the attack surface exposed by Android apps. Also, execute dynamic Java-code on a device, to avoid the need to compile and install small test scripts.
Frida	Dynamic	Android and iOS	A toolkit to inject JavaScript snippets into native Windows, Mac, Linux, iOS, and Android apps to test crypto APIs or trace private application code without the availability of source code.
Radare	Combination of all 3	Android and iOS	A reverse-engineering framework used to analyze and inspect iOS and Android binaries, debug with local native and remote debuggers, perform forensics on filesystems and data carving, visualize data structures of several file types and aid in software exploitation.
QARK	Static	Android	It is short for Quick Android Review Kit. This tool is designed to look for several security related Android application vulnerabilities, either in source code or packaged APKs.

<Table : Security Testing Tools>

1.6 Developer Responsibility in Secure Application Coding

Vulnerabilities in applications are usually the result of mistakes or failure to follow secure coding practices. Vulnerabilities cause risk when they are exploited either intentionally or unintentionally and result in some compromise to a user's data. With proper and thorough code review, these vulnerabilities may be caught during production and prior to release. Also, the risks introduced by coding errors are mitigated to some extent by the architecture of mobile devices. However, some vulnerabilities that are introduced during coding is not recognized or discovered until after the application has reached the marketplace. Even when discovered, applications containing these vulnerabilities remain a risk to the user if the application is not updated or removed. As identified by the studies, the key reasons the security vulnerabilities are introduced during coding or software development phase are as below. [7]

- The carelessness of the developers that they write any sloppy code and left the compiler to run diagnostics.
- The constant demand for novelty in software which is always in the bleeding-edge phase, when products are inherently less reliable.
- The lack of skilled developers and trainings.
- The lack of resources such as code analyzers that can check the security of software and systems

The development best practices and guidelines recommended for developers should ideally reduce or eliminate known vulnerabilities during the build process. Some of these best practices can be applied during development and others should be enforced during the maintenance phase.

Developer training and awareness of secure coding guidelines

Developer training and awareness of secure coding guidelines are vital to secure application development. Training must be conducted before the initial coding phase and should cover common programming errors that introduce vulnerabilities. Application developers should be made aware of and should follow security best practices for secure coding such as those published by Google for Android [20] and Apple for iOS [22]. Awareness of OWASP's list of top 10 mobile risks presented earlier is a good starting point. [12] Mobile application developers should be familiar with the following secure coding guidelines at minimum for secure mobile application development. [19]

1. **Perform secure logging and error handling**

   During development, developers usually maintain commented codes and write logs for different purposes. The following activities related to logging can disclose sensitive information in applications. Hence, the following activities should be avoided during development. [19]

   - Logging to the global log
   - Logging commented code for debugging purposes
   - Performing poor exception

2. **Follow the principle of least privilege**

   By following the principle of least privilege and correctly implementing the permission model provided by mobile OS, the sandboxing and isolation of the application can be ensured [19]. It is always best to minimize the number of permissions that the app requests. Restricting access to sensitive permissions reduces the risk of inadvertently misusing those permissions, improves user adoption, and makes the application less vulnerable for attackers. Generally, if permission is not required for the app to function, it should not be requested. [21]

3. **Validate input data**

Insufficient input validation is one of the most common security problems affecting applications, regardless of what platform they run on. Android has platform-level countermeasures that reduce the exposure of applications to input validation issues, and the developers should use those features where possible [21]. It's important to implement input validations correctly and duplicate client side validations on the server side as well. It's also a good idea to implement security controls for input validations, such as Owasp's Enterprise Security API [19].

Another instant where input validation should be performed is when handling data from external storage as there could be data from any untrusted source. Developers should not store executables or class files on external storage prior to dynamic loading. If the application does retrieve executable files from external storage, the files should be signed and cryptographically verified prior to dynamic loading. [21] In the native code, any data read from files, received over the network, or received from an IPC has the potential to introduce a security issue. The most common problems are buffer overflows, use after free, and off-by-one errors. These vulnerabilities can be prevented by carefully handling pointers and managing buffers [21].

Dynamic, string-based languages such as JavaScript and SQL are also subject to input validation problems due to escape characters and script injection. If the application uses data within queries that are submitted to an SQL database or a content provider, SQL injection may be an issue. Limiting permissions to read-only or write-only can also reduce the potential for harm related to SQL injection. The best defense is to use parameterized queries. [21]

4. **Implement secure data storage.**

Files created on external storage, such as SD cards, are globally readable and writable. As a best practice any sensitive information should not be stored using

external storage as external storage can be removed by the user and also modified by any application. By default, files that are created on internal storage are accessible only to the application. Android platform has implemented this protection, and it's sufficient for most applications. [21] It is recommended to use standard encryption algorithms with strong key values instead of homegrown ones to encrypt sensitive data residing on devices or at the server backend. [19]

5. **Avoid insecure mobile OS features**

Insecure features such as listed below either provided by mobile OSs which goes as a default setting or enabled in the application by the developers can be exploited to extract sensitive data. Such features should be turned off in the application. [19]

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

6. **Encrypt data in transit**

The mobile app should provide appropriate protections for user data in-transit, especially when that data is authentication data, session data, or personal information. New hacking tools have made snooping on unsecure connections quite simple, especially on unsecured Wi-Fi networks. Developers can avoid many of these problems by using SSL/TLS for all communications with your server, as modern back-end providers should have little problem scaling SSL even to a large number of transactions. [23]

7. **Encrypt sensitive user data**

Whenever feasible, the developers should ensure you are encrypting application users' data, especially authentication information like usernames, email addresses,

and passwords. Storing unencrypted data puts both the developer and the users at risk in the event of a data breach. [23]

8. **Protect user sensitive data**

If the application under development accesses personal information such as passwords or user names, the best approach for user data security is to minimize the use of APIs that access sensitive or personal user data. If developer has access to user data and can avoid storing or transmitting it, best is not to store or transmit the data. The best practice is to consider if there is a way that the application logic can be implemented using a hash or non-reversible form of the data. [21]

9. **Protect user's application data.**

It is essential to make sure that when a user logs out of a session using the mobile client, the password changes on the back-end side invalidates mobile clients' current sessions. If the application accesses, collects, or stores sensitive data or is a fruitful target for phishing attacks, consider using two-factor authentication such as confirmation text messages, or one-time application-specific passwords. [23]

10. **Handle authentication with care**

To make phishing attacks more conspicuous and less likely to be successful, minimize the frequency of asking for user credentials. Instead use an authorization token and refresh it. Avoid storing user names and passwords on the device. Instead, perform initial authentication using the user name and password supplied by the user, and then use a short-lived, service-specific authorization token. [21]

11. **Use explicit intents over implicit intents**

For activities and broadcast receivers, intents are the preferred mechanism for asynchronous IPC in Android. Depending on the application requirements, the developer might use sendBroadcast(), sendOrderedBroadcast(), or an explicit intent

to a specific application component. For security purposes, explicit intents are preferred. If intent is used to bind to a Service, ensure that the app is secure by using an explicit intent. Using an implicit intent to start a service is a security hazard because it can't be certain what service will respond to the intent, and the user can't see which service is started. [21]

Use of security assessments tools that can assess application for vulnerabilities

The developers should make use of free capabilities bundled into the application development environment to assess the security of their applications, e.g., the Android Software Development Kit and the Android Lint capability built into Android Studio. [15]. As discussed in section XXXX there are many other security assessment tools available in this space; some integrate with threat intelligence tools to bring more up-to-date information about vulnerabilities or malicious code. [15]

1.7 Problem Statement

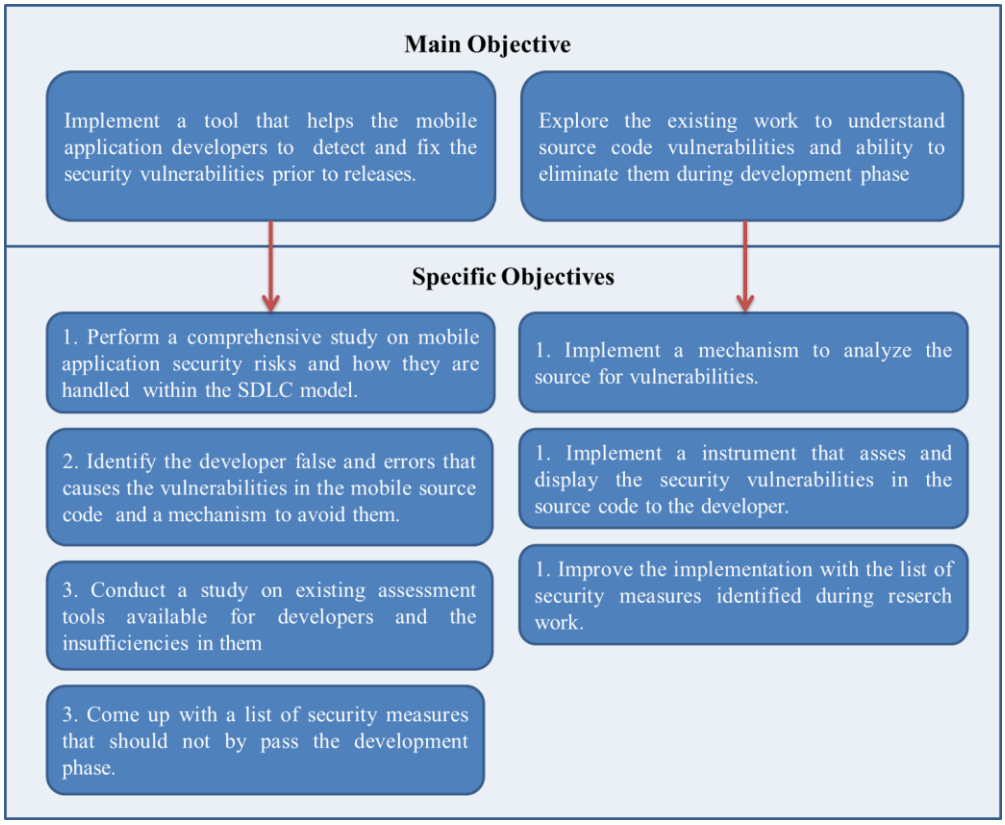
With the growth of mobile application development, it is vital to incorporate security into the mobile SDLC as a strategic initiative. The threat landscape is constantly changing, and with hundreds of millions of mobile apps already launched, it is critical for an organization to utilize static analysis as well as dynamic analysis to get complete coverage of their mobile SDLC. Security assessment can also expose security gaps between project designs and approved corporate policies, which might have evolved during development, or problems resulting from the integration of different modules. [18] Hence, performing static application security testing on the mobile source code will greatly assist developers to ensure that they code securely throughout the development process. There are certain vulnerabilities that are best identified through SAST and may be eliminated early in the process. [16]

The main research problem therefore 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 fixing all security issue tracked, based on secure coding guidelines and pre-defined set of vulnerability criteria. Currently there are several commercial and open source tools which are capable of validating and assessing the security of a mobile application during SDLC. Most of them are used in testing phase when the application has already been built and released to QA. 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 testing and development phases which will ultimately cost to the project in effort and time. Hence, this tool will operate in between the development and testing phase identifying and fixing the security vulnerabilities in the source code prior to QA releases.

In order to come up with a solid solution it is required to explore so many research areas under this research study. The core of this research is based on, secure mobile development which involves many research areas like understanding the security architecture of mobile platforms, identifying and addressing security risks in mobile applications in the source code, identifying the behavior of the build tools and code analyzers used by mobile applications, enhancing the build tools to integrate custom security features, etc. These important research areas will be explored and it is discussed in the literature review section separately.

1.8 Objectives

One of the main objectives of this project is to explore that existing research work to understand the 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 these 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 being built into an executable file. The specific objectives that are expected to accomplish in order to achieve the main objective are listed below.



Initially this implementation will be limited to Android platform as it is open source, freely available and more feasible for implementation. Further on success, this will be extended to cross platform and iOS development platforms as well.

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", 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,", 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,", 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", IEEE Computer Society, pp. 1-3, June. 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,” Proceedings of the 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,” Proceedings on the 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 1999 pp. 103-105.

[10]A. Jaquith, “The Security of Applications: Not All Are Created Equal,” Research Report, @Stake, February 2002.

[11]K. Hoo, A. Saudbury and A. Jaquith, “Tangible ROI through Secure Software Engineering,” Secure Business Quarterly, Q4, 2001

[12]"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, 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.

[18]Tata Consultancy Services, "AddressingSecurity andPrivacy Risksin MobileApplications", IEEE Computer Society, 2012.

[19]J. Burns, “Developing Secure Mobile Applications for Android,” iSEC, Oct. 2008; www.isecpartners.com/files/iSEC_Securing_Android_Apps.pdf

[20]"Best Practices for Security & Privacy | Android Developers", Developer.android.com, 2017. [Online]. Available: https://developer.android.com/training/best-security.html. [Accessed: 19- Nov- 2017].

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

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