**Background**

Mobile application today plays an essential role in the communication process, e-commerce, and entertainment as well as the enhancement of productivity. As more and more smartphones are used to enhance their experience, users of mobile applications are exposed to cyber criminals to a large extent. Personal data like identity, credit card information and medical details are passed to and stored in mobile applications, which makes them an ideal target for hackers. These include; non-reliable storage facilities, poor identification techniques, low encryption standards, and vulnerability to malware. Such vulnerabilities in today’s commercially available mobile applications call for stringent and effective security features in the development process of the applications. Great concern to the safety of mobile applications not only addresses users’ information but does so while building loyalty and guaranteeing adherence to legal requirements.

**Objectives**

The primary objectives of this research project are:

* To identify and compare the security models of Android and iOS to determine the effectiveness of these models.
* To share the key findings concerning authors’ study of frequent misfeatures in mobile applications and resultative consequences for users and businesses.
* To view and discuss various aspects of developing secure code and security measures like authentication and authorization that improve mobile application preparedness.
* To develop a secure Mobile App prototype based on the best and secure practices for Mobile Apps.
* To offer concrete guidance to the developers and organizations to promote approaches for the protection of mobile applications against rising risks.

**2.1 ANDROID SECURITY MODEL**

**Android Permissions Framework**

The Android permissions framework is a vital component for safeguarding user privacy:

* Runtime Permissions: Runtime permission was initiated in the Android 6.0 Marshmallow where instead of being asked to permit the use as the application is being installed, permission is granted when the application is in operation. This approach also puts control of one’s data into the hands of the user so that they can make their own call.
* Permission Groups: Permissions are divided into coherent groups of permissions (for example, location, camera), and it makes interaction with them more natural.
* User Privacy Implications: This approach avoids many applications from gaining permission to access data that is not relevant, thus increases the ability to manage one’s data.

**App Sandboxing**

Android’s sandboxing model isolates each app from others and the system, ensuring robust security:

* Process Isolation: These apps have to open up in individual process which means that each app possesses its own unique user ID (UID) so there is no direct way to attain data.
* Kernel Enforcement: More so, the Linux kernel sustains such isolation with the help of features such as SELinux.
* Security Impact: This means that any possible threats that may lie within an app; will be contained in that app alone, thereby reducing the impact of infected or rogue apps.

**Key Security Features**

* Google Play Protect: Performs a quick scan on applications before installing and in their performing mode. Offers features such as Safe Browsing to help inform users about risky sites.
* App Signing: Protects the application’s authentication as all the applications must be digitally signed. Play App Signing also fortifies security by providing safe control of app signing keys.

**2.2 iOS Security Model**

**App Sandboxing Practices**

iOS enforces strict sandboxing to protect user data and system resources:

* Controlled Access: The Android model has each app running in a sandbox, though permissions are required if one app needs to interact with another or any other resource.
* Entitlement Requests: Entitlements are there and the user need to specifically ask for entitlement which are then checked by Apple.
* Security Contribution: Such seclusion minimizes the chances of any nasty business taking place; the system is protected.

**Data Protection APIs**

iOS uses encryption to safeguard sensitive user data stored on devices:

* File Encryption: This involves use of passcode to authenticate the user while a device key enhances the safety of data to be stored.
* Access Control Classes: Specify when data is available for instance it could only be during times when the device is unlocked.
* Importance: These APIs are very important for applications, which deal with important information, because even in the case when a device is stolen or hacked, important information will always remain safe.

**Secure Boot Process**

The iOS secure boot process ensures that only trusted software runs on the device:

* Chain of Trust: Starting from the physics Root of Trust up to the Kernel of iOS, every step ensures the validity of the next.
* Code Signing: No other code can be run on the iPhone; all programs need to be signed by Apple.
* System Integrity: This process served as the basic for device security, protecting it from boot time attacks.

**3.1 OVERVIEW OF MOBILE APP ARCHITECTURE**

Mobile application architecture means the structure and a set of patterns that regulate the external and internal interaction of the components of a certain mobile application as well as their organization. This architecture provides base or framework for the creation of effective and sustainable mobile apps.

**Importance of Mobile App Architecture:**

* Performance: On this basis, architecture guarantees effective application of resources that are available in a specific system and ensures the convenience of using the developed application by the target audience.
* Scalability: A scalable architecture of an app serves to accommodate additional users and operate additional functions with less effort.
* Maintainability: The nominated solution provides the best guarantee for reducing the amount of time that needs to be spent on future app updates and bugs’ fixing due to a clearly defined architecture.
* Security: Good architecture incorporates layers of protecting data, communication as well as user interaction to limit the exposure.

**3.2 Types of Mobile Application Architecture**

**Monolithic Architecture:**

A monolithic architecture covers one large application, and all its aspects (UI, business logic, data access) are integrated.

**Advantages:**

* Can be implemented with discrete, easily managed changes which maybe suitable for low function applications.
* It is easy and efficient to write and test code in a single place that will be used across multiple components.
* Earlier, external dependences beget complexity and that is the reason why they are low.

**Disadvantages:**

* The scalability must be done as the application develops, which can result in page bottlenecks. The decision-making process is also important because it can direct the user to an unintended desired state.
* Any modification made can affect others and complicate maintenance work.
* It has been observed that constant addition of fresh plugins means that the entire app has to be redeployed which in turn may cause a break in service.

**Microservices Architecture:**

Microservices architecture splits the application into small independent services and each will handle specific functions. These services interrogate each other through application programming interfaces, abbreviated as APIs.

**Advantages:**

* Scalability: All of them can be distributed across services in such a way that resources can be kept at optimal levels.
* Flexibility: Apparatus’s can develop different services simultaneously in parallel thus increasing the process speed.
* Resilience: The breakdowns that occur in one service do not affect all the application services.
* Continuous Deployment: Services can be updated themselves without having to update the entire system all together thus decreases time spent on damages.

**Disadvantages:**

* Complexity: The management of many services is a complex task that needs efficient tools and knowledge.
* Inter-service Communication: May cause the problem of latency and dependency.
* Increased Resource Usage: Many services require more underlying infrastructure required which in turns costs a lot.

**5.1 Methods for Mobile Application Hardening**

Mobile app hardening is a process of securing a mobile applications against reverse engineering or modification and protection against malicious use.  
  
**Code Obfuscation:**

* Translates the app’s source code into an obscure form which is not easily comprehensible as long as the convenience of an application is maintained.
* Reduces the ease that an attacker gains when trying to reverse engineer the app.
* Android followed by iOS instrumentations are generally used ProGuard Shield and Swift Shield respectively.

**Encryption:**

* Provides privacy and security of information when in transmutation and warehousing.
* Data-in-Transit: Securing of the application communication with the servers can be done using TLS (Transport Layer Security).
* Data-at-Rest: Secure local data that has not been synced with AES or Advanced Encryption Standard.
* Utilize strong key storage technologies utilizing trusted execution environments, Android Keystore, and Secure Enclave for an organization.

**Secure Storage Solutions:**

* Android: For database encryption use Encrypted Shared Preferences and for SQLCipher.
* iOS: Use Keychain Services for storing of secure credentials and tokens.
* Backend: Use hases keys to make secure synchronized connection between the specified application and the cloud storage.

**5.2 Secure Coding Practices**

Security coding must be done to minimize leagates and guarantee a safe application.  
  
**Input Validation:**

* This can be done to avoid injection attacks like SQL injection and or cross site scripting.  
  In parameterized queries, use regex in order to filter out wrongful formats of the input data.

**Output Encoding:**

* Prevent XSS by encoding data prior to when it is displayed on the webpage of the application’s user interface.
* In the integrated environment shown above ensure that special characters in the output are treated as strings not as code.

**Secure API Usage:**

* Encrypt an application with SSL and/or TLS, apply OAuth 2.0, JWT for API endpoints.  
  Set up rate limits zad an API being abused and DoS attacks.  
  Reduce read access to the data by exposing APIs only to the required data.
* Avoid Hardcoding Sensitive Information:  
    
  For example do not keep your API keys, or login credentials within the code or within the repository.  
  Rather than that, use environment variables or encrypt the configuration files.  
  Regular Security Testing:
* Perform a static code analysis, a dynamic code analysis, and penetration testing to find the dangers.  
  You’d better try OWASP ZAP or Burp Suite to organize attacks and define potential security dangers.  
  Use Secure Development Frameworks:
* Use the library and frameworks designed to help to provide security to the communication ciphertext, and validation of the input data.

**6.1 Introduction of Authentication Protocols**

Using authentication protocols a secure authentication of users in a mobile application is achieved. The following frameworks are widely used:

**OAuth 2.0:**

* An authorization structure that has gained popularity in the authorization process to enable the third-party applications to interact with the user resources without necessarily compromising on the credentials of the user.
* Key Components: Authorization server, resource server, client and resource owner.

**Advantages:**

* Supports all forms of grants like the authorization code grant, client credentials grant. Furthermore, it replaces credentials with access tokens to improve security at some extent.
* Common Use Cases: Social account sign-up, APIs.

**OpenID Connect (OIDC):**

* A layer of identity based on OAuth2.0 for authentication and login-sign on.

**Key Features:**

* Returns an ID token that contains some information of the user.
* Provides security with the help of JSON Web Tokens (JWT).
* Advantages: Speeds up the authentication process with integrated get user profile procedure.

**Security Assertion Markup Language (SAML):**

A framework that is in simple terms XML based framework for exchange of the authentication and the authorization data between the IdPs and the SPs.

**Key Features:**

* Uses assertion statements for transferring authentication information, which is an assertion.
* They are best suited for use in enterprise since they are designed with complex windows or door operating systems for building and management of premises.

**Advantages:**

* Allows users to use one service to log into other services; the services can be owned by different organizations.
* It integrates well with the current enterprise identity systems especially with high compatibility.

**6.2 Multi-Factor Authentication (MFA)**

MFA enhances security by requiring users to verify their identity through multiple factors:

**Types of MFA Methods:**

* Knowledge Factors: Information that the user possesses (e.g. passwords, Personal Identification numbers – PINs).
* Possession Factors: A token which the user possesses (OTP through SMS, a piece of hardware).
* Inherence Factors: At least one function that shows the chosen concept describes something the user is (biometric login such as fingerprints or face identification).

**Common MFA Implementations:**

* Time-based One-Time Passwords (TOTP): One time tokens which are created using applications such as Google Authenticator.
* Push Notifications: You can approve or deny login requests through a mobile application.
* Hardware Security Keys: Examples of such devices encompass the use of YubiKey and FIDO2 since they offer a more secure way of processing touchless authentication.

**Benefits of MFA:**

* Tackles the problem of having a credential that can be attacked through a technique such as phishing or brute force attack.
* Allows access only to the person who possesses the right combination between two or more types of factors.

**6.3 Secure Session Management**

While session management guarantees users with authenticated sessions, users have authorized access to resources.

**Best Practices for Secure Session Management:**

**Session Tokens:**

* Use a token that is generated through standard 8 character long alphanumeric strings or use cryptographically sound tokens such as JWT or opaque tokens.
* Store tokens securely on the device (on Android it’s SharedPreferences, on iOS it’s Keychain).

**Token Expiry:**

* To ensure that tokens are not misused, the token lifetimes should be very short.
* Refresh tokens can be employed for sustaining user sessions with reasonable security.

**Secure Transmission:**

* Never send tokens over HTTP as they can be intercepted easily.

**Session Timeout:**

* Timing out users after certain time to minimize exposure - log users off when they are idle for some time.

**Token Revocation:**

* Include ways of revoking tokens when the client logs out or when someone hacks his/her password.

**Additional Security Measures:**

* Device Binding: Ensure that connections are tied to what is actually them device to prevent re use of the token on other devices.
* IP and Location Monitoring: Notify events like login from a different geographic location.