HOW TO:
Privacy Aware Mobile Application Development

Research Performed For:

Privacy Rights Clearinghouse

https://privacyrights.org

Technical Research, Analysis, and Documentation By:

Craig Michael Lie Njie*

Version History of this Document:

- First Public Release: Version 1.0, Released July 15, 2013
- Most Recent Update: Version 1.1, Released August 12, 2013

* Mr. Lie Njie has over 20 years of professional technical experience, and in the last four years has designed, developed, deployed, and performed technical analysis of a variety of mobile applications for both Apple's iOS and Google's Android. He is Founder and CEO of Kismet World Wide Consulting: http://www.KismetWorldWide.com

Table of Contents:

1. Executive Summary: A privacy HOW TO for mobile app developers

This document identifies and describes several rules and problem-solving approaches to building mobile apps that focus on protecting users' information privacy. This HOW TO guide is part of a larger report in which we analyzed how a representative range of mobile health and fitness apps operate technically.

We analyzed how a representative range of mobile health and fitness apps operate technically. After reviewing privacy policies, we installed and used the apps. We looked at what data applications stored locally on the devices, and what they sent over the network. We noted how closely an app's actual functions mapped to the description of the developer's information practices in any published privacy policy, and we identified technical practices that pose privacy risks.

Since most health and fitness applications require continuous input of user data to fulfill their purpose, there is an inherent risk of harm if that data is compromised. Developers can modulate the risk by how they collect, store and transmit data, but nevertheless, the more user data an app collects, and the more sensitive that data is, the greater the privacy risk to the user of that app.

The good news is that there are best practices for building a high degree of privacy protection into mobile applications. This document highlights what we believe are the most useful of such solutions.

The bad news is that **none of the apps we evaluated** followed all the best practices we describe in this paper. We do not believe this widespread lack of technical measures to insure privacy protection is due to malice or ill-intent, but rather because **mobile app developers generally do not know how to build privacy into apps** that collect, store or transmit privacy-sensitive user data.

This document is a summary of best technical practices for mobile app developers to insure the highest levels of privacy protection.

Although this HOW TO is intended for developers of mobile health and fitness apps in particular, the advice it contains applies to mobile apps in general.

2. Intended audience

While our focus is on health and fitness apps, mobile application developers in general can benefit from the privacy-protecting suggestions in this HOW TO. Although it is written for a broad audience with a variety of backgrounds, it assumes a base level of technical knowledge about how mobile apps are built, how they collect and store data, how they communicate data over the Internet,

and how third-party tools and services like advertising networks and analytics services are integrated into a mobile app.

This document will also be useful for product managers to use as a checklist to review the code of their mobile apps.

3. Overview: Primary technical privacy risks to users of mobile apps

3.1. Unencrypted network communication: Always use HTTPS

Many apps send personal and sensitive user information over unencrypted (HTTP) connections. Developers should insure that all network communication is encrypted. The easiest way to do this is to always use HTTPS; never use HTTP to transmit data from the app to a server on the Internet.

3.2. Advertisers

Apps frequently share personal information with third-party advertiser networks, either in keywords sent to advertisers to target their ads, or embedded in URLs sent to advertisers.

Ideally, developers should avoid using ads in their apps. If the developer's revenue model requires ads, however, network communications between the app and advertisers should not include any personal user information and should always use HTTPS.

3.3. Analytics services

Many apps share users' personal information with third-party data analytics services. Developers should avoid using third-party analytics if possible; if not, they should withhold personal information from analytics companies and fully anonymized all data shared.

4. Privacy checklist

4.1. Limit the collection, storage, and network transmission of user data

Don't collect, store, or transmit over a network any data that is not required by an application's core functionality.

4.2. Notify users of all data collection and explain its purpose

Inform users clearly what data is being collected, when and for what purpose. The best way to do this is at the time of collection through a contextual pop-up notice.

If real-time notification is not possible, be sure to describe data collection practices in detail in the app's privacy policy. If the privacy policy is overly lawyerly, call out information about data collection and uses, and user access and controls, in a consumer-readable FAQ.

4.3. Encrypt all network communication with SSL or stronger

EVERY single network connection should be encrypted. The simplest option is to use SSL connections for low-risk data. This includes **all** connections to the developer's servers and to third-party sites like advertisers and analytics.

4.4. Use high-grade encryption for users' personal information

Apps that use, collect and store, personal information should protect it with stronger encryption than just SSL. Even when data is encrypted, always use SSL connections to transmit it.

There are several encryption SDKs and tools available for use, many for free, some as part of the iOS and Android SDKs—too many options to list here. You'll need to find the best tool for your particular need and budget.

4.5. NEVER send user information in clear text

All network communications should be sent encrypted over SSL. This includes connections to developers' servers and to third-party sites like advertisers and analytics.

4.6. Always use POST, never GET

Never send data as part of a GET argument, for instance:

```
http://site.com/upload?user=MyName&email=me@myemail.com&password=1234
```

This data will still be seen in the clear as it travels over an unencrypted HTTP network connection, the information is usually stored unencrypted in the traffic logs on the remote web server(s), and all local caches stored on the device will hold that URL data in the clear putting users at risk if their device is compromised.

Instead, send user data via another mechanism, either via POST over HTTPS (SSL-encrypted network connections), or something you've built using custom encryption and/or data obfuscation. In all cases, make sure that all user data sent over the network is encrypted from end to end.

4.7. Salt and hash all passwords before sending or storing

Never store or send a password as cleartext; always salt and hash the password when the user enters it and only store and send the resulting hash value. Should you ever need a password entered (e.g. to recreate the hash) ask for it and then store the new hash.

Never send a password in cleartext as part of a GET argument (even over SSL) or as a POST argument over HTTP.

4.8. Do not expose information in URLs – obfuscate page names and data

A number of apps we looked at expose private data in the URL itself, which is sent in the clear over the net and stored in local web caches on devices. For example:

```
http://site.com/aids/recently_diagnosed/support_in_new_york.html?diagnosed=Jan2013
```

Notice that this URL exposes the following personal and sensitive information:

- the user likely has AIDS,
- the user is probably in New York, and
- the user was probably recently diagnosed with AIDS in January of 2013

Compare with this URL:

```
http://mysite.com/get_page.cgi?page_id=141245?session_token=321
```

It is more difficult to gain any insight into the user from the second URL, which obfuscates both the page content and the user's personal data.

If it's necessary to send something like the date of diagnosis, do not send it on the GET request as in the first URL. Instead send it as a POST argument over an encrypted HTTPS (SSL) connection.

4.9. Protect against URL replay attacks by using single-use or expiring URLs

No amount of obfuscation of the URL can protect against a URL replay attack where an attacker simply opens the URL in their own browser to see the content.

To protect against this, make sure the URL is not reusable – i.e., it cannot be used by anyone but the user at the time the user requests it.

There are several technical solutions to the problem, but the simplest is to expire the URL so that it doesn't work after the first time a user loads it. Another solution is to use a cookie to store a one-time-hash of the session, and deny loading of the page without both the URL and the cookie.

Each technical solution to protecting against a URL replay attack has its pros and cons. Developers should choose the one that meets the specific requirements of both the app and their server setup, while maximizing the privacy and security protection of all user data.