Security of Mobile Tracking Systems

Dan Mihai DINU
IT&C Security Master
The Bucharest University of Economic Studies
ROMANIA
dan@aplicatiimobile.com
http://www.aplicatiimobile.com

***Abstract**:* This document will describe an implementation of a closed Mobile Tracking System which can be used with Android devices but can be extended to any platform. A user is able to setup the system and permanently receive location updates from his tracked devices. The system is completely secured and it sends transparent location updates. It can be used in many ways. As a Proof of Concept the system will be implemented over a Car Fleet Tracking example.

***Key-Words**:* Secure Tracking Systems, mobile, cloud

1. Introduction

In less than three decades, the mobile phone has gone from being a status symbol to being a ubiquitous technology that facilitates almost every interaction in our daily lives.
As adoption of advanced mobile devices such as smartphones has exploded in recent years, consumers have grown increasingly comfortable using their phones for anything from entertainment and leisure use to financial transactions and business use. The world is going mobile and this trend is due to the increasing number of ideas and developers working in this new area.
Owning a smartphone in these days became a must; with it you're permanently connected.

2. Motivation and problem formulation

Mobile development has now no limits. Millions of applications exist and may be downloaded.
Location based applications are growing considerably and everything is moving around these type of services from search engines to friends locator applications. Why? It comes natural to search for people, places, attractions, interests nearby. With a mobile device, it's easier to find what you're looking for. Location-based services have a huge potential, more and more companies are approaching this this area. Take for example services and applications provided by Google like Google Local, Google Places, Google Maps, Google Latitude and the new Google Glasses. Same for Facebook: Facebook Graph Search, Facebook check-in and the list continues.
This paper will present a generic tracking system which, as an example, will be customized for tracking cars from a car fleet. The base algorithm remains the same, thus by customizing the user interface, and data that is extracted from the application it can be implemented/extended to different tracking systems.
There have always been problems when companies lending cars to clients or employees. Common issues are that employees use company cars also for personal use when they should only use them in business scope, Or, in the case of a rental car company it's of common sense to know where your car is located and what is the client doing with it. Does he break any rules?
It would be a lot easier for car fleet owners to always keep track of their cars, know at any point where their car is located, see their location live on a map, always be aware on how the car is used. A whole lot more exciting functionalities can be added to this example, some of

ISSN 2067 – 4074

them will be presented further in this article
The basic idea is that building a system which can transparently send updates about location from any mobile device back to a user will take tracking to a new level.
Considering these fact the problem sounds in the following way:
Build a mobile tracking system based on device location in order to perform a silent, background tracking of any Android device consuming at least resources as possible.
Tracked Android devices must sent transparent updates about their current location to a client application. The client application must interpret and display live location data received from the tracked devices.
The entire system needs to be closed to the external world. In order to use this tracking system, build secure authentication logic so that only eligible users may access tracking data. A user must be independent and not know any information about any of the other users. The system also needs to be closed at user level. One must be able to obtain data only from the tracked devices that he configured.

2.1 What other apps are out there?

No application having the functionality mentioned above is implemented yet. There are apps available in mobile stores that use the users GPS or Network location to provide information or services related to your coordinates. Here are some examples:

- Social Apps (Google Latitude for Android, Find My Friends for iPhone, WayApp, Badoo, different dating apps) – these apps have different features but the basic functionality is that they let you see the current location of your friends or family, the distance away from each one of them, their location on a map, the fastest way to get to them and so on.
- Taxi Apps (Star Taxi, Speed Taxi) – implemented over the same base idea. These apps let you order a taxi from your mobile phone. When you press the "Order" button all taxi drivers nearby get notified on their smartphones about the order. Whoever accepts the order faster gets to pick you up from the location specified.
- Google Local – based on the user's location Google Local suggests restaurants, cafes, bars, attractions, hotels, ATMs, gas stations, basically anything that you can think of. The app shows the distance between you and the Point Of Interest and you can do many things such as: rate the place, comment on the place, check-in in a place, get directions to it by ground, by car.

2.2 What will the application do?

The system can be installed on any group device that is running Android.
There are two parts, two different client-side Android applications that integrate and make the system work. These two will be called *Fleet User* and *Fleet Car* from now on.
Fleet Car is the application that needs to be installed on the devices that will be tracked down.
Fleet User is the application that needs to be installed on the user Android device. This will help the user receive information about the tracked devices (cars in this case).
The tracked devices which have *Fleet Car* installed will send transparent location updates to a third-party server. Once the server receives new locations for a device it will push this information to the *Fleet User* app, which will present the information to the end-user.
The end-user will be able to receive live information about what he's tracking down and permanently see his tracked devices location.
This workflow will be presented in more detail in the following chapter.
Both *Fleet user* and *Fleet car* contains different options that may be customized by the user like location update interval.

3. Problem Solution

The architecture and workflow consists of two main parts and a third party server through which the communication

between the mentioned two parts is made.

1. The Client Side (further referenced as *Fleet User*)
2. The tracked device side. (further referenced as *Fleet Car*)
3. The Third Party Server

The app behavior is different for each part. The two parts communicate through third-party web-services and server.

3.1 The Tracked Device Side/Fleet Car

The Tracked Device Side must have a smart device equipped with GPS and internet connection that can support the *Fleet Car* app. This app needs to be installed on the devices that need to be tracked down.

When the user launches the app the first time he is asked to login with his/hers Facebook account. Once the user is authenticated and authorized through Facebook API the user will be redirected to the Application main Graphical User Interface. He can enter here the tracked car Name and start/stop the service (explained below) that will run in the background forever.

Fleet Car will run as an android background service in order to permanently and transparently send location updates to the server. Once the service is started in will continuously send updates at the specified interval of time. A database entry with the Facebook User ID and the Android Device ID as the Primary Key will be updated when new location coordinates are sent to the third party.

The latitude and longitude found in the database are always up to date and are always the most accurate. The algorithm behind scenes sends update messages to the database only when it's needed in order to prevent battery consumption and internet traffic usage. For example, if the user doesn't move then there's no need to send location updates or if only 10 seconds passed since the last update then the user could slightly have moved a couple of meters so an update won't be necessary either. An optimized algorithm takes the decision if a location is more accurate than another one.

In order to receive location updates from the tracked devices the user needs to install *Fleet User* application on his Android device.

When the application is launched the user is asked to provide his Facebook credentials in order to be able to receive information about his car fleet. He needs to login with the same Facebook account that he used when he configured his tracked devices. This is a closed system and the only the user and nobody else is authorized to see its tracked devices locations.

All the cars are presented in a List View. If any car entry in the list is pushed than the car will get displayed

The device connects and listens to the same third-party server as the *Fleet Car* service. After login the *Car User* application will retrieve the latest locations of all cars. After this primary update it will wait to get notifications from the server in order to update the information.

The third party server consists in web-services and NoSQL database in the Google Cloud. The role of the server is to securely perform communication between the tracked devices and the user. The server receives update messages from the tracked devices and updates entries in the database. Once a new location is received it also pushes this information to *Car User* app via Google Cloud Messaging system.

4. Security Impact

When data is exchanged via Internet there always are security breaches. The communication needs to be secure.

4.1 The tracked device party

The user needs to use his Facebook Account in order to login within the app. Facebook login is secured via https protocol and different Facebook internal security walls. Facebook API securely authenticate and authorize the user if this operations are successfully completed a session key and access token are received and persisted locally on the mobile device. The login part is completely secured via Facebook services. If any exception is raised during the login operation then the

ISSN 2067 – 4074

user will not be able to use the application.
Communication between the tracked device and third-party webservices is done via http because of cost considerations (Google App Engine will be used as a third party cloud provider). These packets may be intercepted and the attacker can find the webservice address and information pattern that he needs to send in order to send fake requests or compromise data. This can be avoided by encrypting the data using a third party library. By doing this spoofing, DdoS attacks can be reduced.

4.2 The Client Party

The client party (*Fleet User*) behavior is similar to the End-User party. The smart device connects via Facebook account to the app in the same manner that the tracked device party does and then communicates with Google Cloud third party services.

4.3 The Third Party

The role of the third party server is played by Google App Engine. Google App Engine offers cloud services such as software as a service, platform as a service, infrastructure as a service, database and storage services in the cloud.
Google App Engine was chosen because of the free services that it offers in some limits.
Webservices are deployed directly in the Google Cloud via Google App Engine. These webservices alongside with the data stored in the database (personal info and location of users mentioned in section 5.1) run 24/7 and they should be completely secured.
Unauthorized access to data and code are guaranteed by Google so there are no worries in this direction.
Possible attacks which need to be handled can be DDoS attacks and spoofing attacks.

DDoS attacks

In computing, a denial-of-service attack (DoS attack) or distributed denial-of-service attack (DDoS attack) is an attempt to make a machine or network resource unavailable to its intended users.
Reading and writing from/to the database costs The free services that Google offers include some read and write limits to the database (50 000 reads and 50 000 writes per day, extra costs money). Nobody should know how to access the webservice because there were some measures taken, which were presented above: communication between parties is encrypted. Data validation is made on server-side before being processed.
An attacker can see the address of the web service but would not know what parameters he needs to send in order to generate, for example, 50000 read requests and compromise the server.
But what if he finds out the right parameter data-types, order? Then the attacker could easily generate 50000 requests in just one second and compromise the whole application workflow. Also the user can send infinitely request, overload the server and make it unavailable for lid users.
But that's why Facebook Login was chosen in the first place. As specified above, when the user logs into the application Facebook sends back an access token. This can be used on the server side to verify if the request is valid. Alongside with the location coordinates, *Fleet Car* sends to the server the Facebook user id, the android device id and the access token.
We know that updates are performed at a minimum interval of 1 minute. So if a location update request is received from the same device in less than one minute then this may be a DDoS attack tentative. If more requests are received by the server in less than one minute from the same device then further requests from this device will be ignored.
If the request is valid then the server first checks the access token validity. If everything is ok then it updates the database with the new location. This mechanism will avoid any type of DDoS attacks.

Spoofing attacks

A spoofing attack is a situation in which one person or program successfully masquerades as another by falsifying data

and thereby gaining an illegitimate advantage.
Theoretically, spoofing can be done in different ways. For example, if an attacker can find a way and pretend that he is a legitimate user he can send fake information to the server which is an undesired scenario.
In our system this is avoided through Facebook authentication and authorization. When an update request reaches the server side the server checks with Facebook if the user access token is valid and if the user is really who he claims to be.
.Another spoofing method in this case can be GPS Spoofing. A GPS spoofing attack attempts to deceive a GPS receiver by broadcasting a slightly more powerful signal than that received from the GPS satellites, structured to resemble a set of normal GPS signals. These spoofed signals, however, are modified in such a way as to cause the receiver to determine its position to be somewhere other than where it actually is, specifically somewhere determined by the attacker. Because GPS systems work by measuring the time it takes for a signal to travel from the satellite to the receiver, a successful spoofing requires that the attacker know precisely where the target is so that the spoofed signal can be structured with the proper signal delays. This is a very hard to accomplish task but yet possible. For example, an attacker could send fake GPS signals to the tracked device. The server will receive a fake location and provide user device with inaccurate information. There is no way to counter attack this issue.

Unauthorized use of phone
Another security breach can take place if an unauthorized person uses your phone This is a common security breach for all applications. This issue can be workaround by forcing the user to add a security code to unlock the phone. Some examples of apps that do this are iBanking applications and VPN applications.

5. Mechanism and data flow analysis

Communication between *Car Fleet* and *Car User* applications are done via the third party server.
The third part server is located in Google Cloud. On the third party server more web services are deployed which can be accessed via http requests. The same server also holds a Datastore where locations for each device are being stored.
Two main webservices are deployed on the server: *UpdateLocationServlet* and *RetrieveCarsServlet*. Both of them can be accessed via http requests.
UpdateLocationServlet main role is to update tracked devices location into the Database(DataStore).
RetrieveCarsServlet role is to return all cars location information for a specific Facebook user.
The mechanism and data flow that makes everything work together is presented below:
Starting from the tracked device, meaning the *Fleet Car* service, the *LocationListener* implemented in the service starts listening to the GPS sensor. Once it receives a new location from the GPS sensor it checks to see if the location is more accurate than the last one.
If the location is more accurate it starts an asynchronous task that runs as a background thread which builds a http request containing: android device unique ID, Facebook user ID, Facebook access token, latitude, longitude, car name, and location date(year, month, day, hour GMT difference: Eg: 2013-05-04T13:43:46.166+01:00).
Once the http request is build it is sent to *UpdateLocationServlet*.
When the request is received on the third party Webservice security checks are performed. If the user is indeed legitimate then the location is updated in the Datastore. New data is pushed to *Fleet User* application containing the new location for the specified car via Google Cloud Messaging(GCM). *Fleet User* receives the new information and updates the GUI accordingly.
RetrieveCarsServlet is used when *Fleet User* app is opened and has no data

ISSN 2067 – 4074

whatsoever. In order to get the data that needs to be displayed to the user the app starts an asynchronous task which builds an http request containing the user's Facebook ID and access token. When the request reaches the webserver the user is verified to be legitimate. The Webservice then reads from the database (or more often from the MemCache which will be explained below) and returns the data to the user in JSON format.

A MemCache mechanism is implemented between the webservices and the actual database. Because requests come very often to the webservices they need to work with the database very often. Read and writes directly to the database cost time, performance and money. Thus, a MemCache mechanism is implemented. The MemCache acts like a wall in front of the database. For example, when *RetrieveCarsServlet* receives a request to retrieve all cars of user X it first looks for the cars in the MemCache. For the cars found here there's no need to read the data from the real database. The MemCache is permanently up to date and contains the latest data.

6. Conclusion

Many features and enhancements can be added to the system in order to offer different services in a user-friendly way such as presenting a car average speed, maximum speed, number of kilometers, route that it follow and so on.

This system can be used in various ways and the Car Fleet example was one of them. This can be implemented at different levels and can be applied in different domains. For example this system can be used to track down people or to track your own device in case it gets stolen.

Acknowledgment

Parts of this research have been published in the Proceedings of the 6th International Conference on Security for Information Technology and Communications, SECITC 2013.

References

[1] E. Kochi, "How the Future of Mobile Lies in the Developing World", *TechCrunch*, May 2012

[2] R. Rogers, John Lombardo, Zigurd Mednieks,Blake Meike, *Android Application Development*, O'Reilly, 2009

[3] Ed Burnette, *Hello Android,* Pragmatic Programmers, 2012

[4] Facebook, *Facebook SDK for Android*, 2013

[5] Google, *Android SDK,* online, http://developer.android.com/sdk/index.html