**Thesis Idea**

**Goal:**

1. Ensure that only legitimate users in target areas are eligible to register.
2. Easy for users to register that as many participators as possible can take part in the task.
3. Lower the burden of server infrastructure.

**Aspects of work:**

1. Registration methods. This could apply to all kinds of registration. Referring to Google might help.
2. Authentication method from server side. How to tell human apart from bots.
3. How does adversary evade rooted devices(and the countermeasures to them).

Client side:

Most of the work like detecting sensors or other peripherals are not so difficult. However, have to dig into rooting detection, like mentioned in the thesis. Maybe can detect rooted info from kernel layer with C++.

Sever side:

Spring boot and database implementation. Learn how to implement backend server in one month. General idea is: mobile side sends its testing requests and server sends back the challenge. Between client and server, the message should be secured and cannot be spoofed easily.

We want to separate low quality eligible users from strong malicious spammers.

How could malicious users hide their real identity?

**Root, IMEI serial number .... (supplement)**

Each of them we should have the countermeasures. Could be from both client side and server side.

CAPTCHA farm

Challenges are redirected to developing countries where cheap workers could be found. They solve the CAPTCHA challenge manually and send the response token back to the bot developer.

Depending on the responding time, we could generally determine whether it’s a bot or CAPTCHA or a normal man.

About CAPTCHA

As far as I know, it’s not possible or pretty hard to fake sensor data (e.g. temperature, accelerometer, etc.) Therefore if we use sensor to do the CAPTCHA challenge, we just need to guarantee the device is not an emulator.

Previous thesis is doing CAPTCHA challenge locally but it’s not really practical to do it locally considering the storage CAPTCHA may take.

I just realize that it’s pretty hard to evaluate the system security level especially botnet detection.

Implementation of the CAPTCHA:

Server generate picture or some random sequence for frontend to generate the captcha test. Picture can be transformed into String format.

**Literature Overview**

**Recruitment Framework for Participatory Sensing Data Collections**

Mobility models:

**Location Summary for Personal analytic:**

Consecutive location points within a certain time period are aggregated

into clusters. Can be used to locate user.

(Would this apply to the separation ?)

**Location Prediction and Location based Services:**

These models take a very microscopic view on mobility, concentrating on determining which “cells” a user might travel based on transition patterns from previous cells, time spent in the current cell, and speed/trajectory information

Related papers:

16. Bhattacharya, A., Das, S.: LeZi-update: an information-theoretic approach to track mobile users in PCS networks. In: Proceedings of Mobicom, pp. 1–12. ACM, New York (1999)

17. Soh, W., Kim, H.: Dynamic guard bandwidth scheme for wireless broadband networks. In: Proceedings of Infocom, pp. 572–581. IEEE, Los Alamitos (2001)

**Android Rooting: Race between detection and evasion**

Xposed framework, Java hook to achieve root evasion.

Hook the detecting methods, and execute any code. However only from Java level, very hard to evade from native layer.

Deploy root detection from both Java and native layers

**Android Rooting: Methods, Detection and Evasion**

Mention 7 detection methods(Mostly from Java level). However all of them can be evaded.

Checking common rooting apps: hook package checking methods and hide those sensitive package symbol.

Files or file path detection: hook methods and return fake path detection results.

Build tag detection: usually build tag retrieve from *android.os.build* would be “release”. However this can be modified by Java reflection. Not to mention release tag can also appear in some rooted image.

Check System Property:

....(Other three detection methods)

With hooking method, Hook API would appear at the top of the calling stack of the exception, which indicates that the device must be rooted. (This could be hided if root cloaker tamper with Java runtime

In addition, usually all detection are stateless, which means only return values are required. This makes convenience to root cloakers since they only have to hook the function calls and return fake results. If some shell or pipe which would not trigger root evasion, it would be harder for cloaker to evade. In such condition cloakers have to examine both input and output of the pipe.

(Maybe instead of directly call function Runtime.exec(‘su’), it’s better to first build a shell process using ProcessBuilder.start() to generate a pipe)

**A New CAPTCHA Interface Design for Mobile Devices**

Published back in 2011, therefore a little bit early. Only mention distorted text CAPTCHA and I think there was not sth like ReCAPTCHA back then. Make use of similar animal models to generate a CAPTCHA image. However so far, I think this could also be easily solved by machine learning to tell the difference between animals.

**BeCAPTCHA**

Published in 2015, relatively new paper. Use data set, containing human samples and synthetic ones to train Generative Adversarial Network.

(The result of the research is a little bit difficult to understand)

**Gesture based CAPTCHA**

Distorted text recognition is very similar to Google’s ReCAPTCHA. To make it more sophisticated and user friendly to mobile users, they implement gesture-based CAPTCHA which require human to interact with mobile devices according to the instructions stated in the “distorted sentence”

Comparing to ReCAPTCHA, the success rate is higher, however cannot be proved more strengthened comparing to ReCAPTCHA, not to mention v2 or 3rd generation.