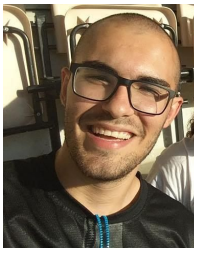
Remote Child Locator

DoTrackMe

SIRS Alameda Group #20

73422	Kevin Santos	
79710	David Cardoso	
80937	Guilherme Quintino	

Introduction

Children are amongst the most vulnerable people on the planet. Also, a great portion of them use smartphones. We constantly worry about their whereabouts, but when you try to call them, they don't pick up, either because they're in class or simply don't notice. And then we have that tingling feeling on the back of our mind saying: "something is wrong" ...

What if we could know if something is wrong by having their location on our smartphones? Then, you would know where they are supposed to be at (like school or the library) instead of some friend's house or in the middle of a shady neighbourhood.

Security Requirements

(in green are the ones who were met)

Redundancy: App will not be shutdown. Multiple ways of tracking the smartphone.

Authenticity: Multiple level login layers. Local, remote and confirmation on add device.

Freshness: The use of nonces on register/add device. Feedback on deprecated data is shown in the app if server/tracking not available.

Integrity: Hashing messages and encrypting them for them to be compared at destination.

Confidentiality: Messages from/to the app are encrypted

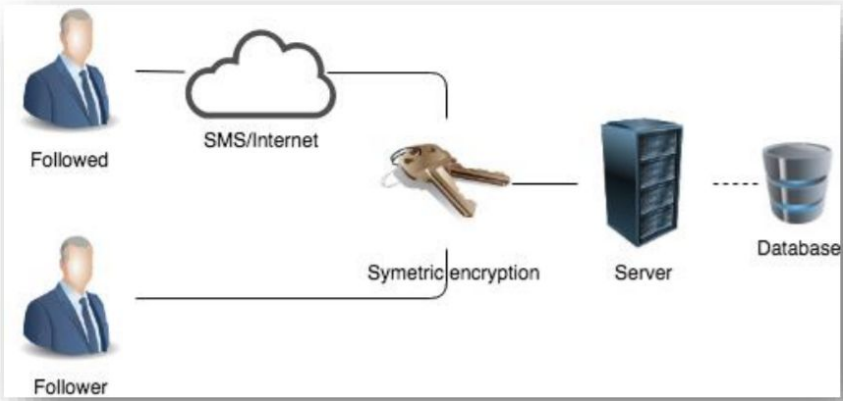
Non-Repudiation: Usage of certificates/signatures will ensure that all messages come from the server and the other way around.

Proposed Solution

There will be at least two users and a server in play. The server has a connection to the databases where the login data and the list of users who are tracking other users. Ideally there would be two databases for each data types, but due to simplicity, at the moment only one will be set. Users register through the server, and the server will use the mobile network SMS to validate the registration. Also, SMS will be used to send tracking data in case of no internet connection.

The database will be storing passwords (hashed with salt) in order to protect them from unauthorized access. Reauthentication will be necessary every time you update the tracking information (valid for 24h). The app on the smartphone is protected by a local 4-8 digit pin number (locally hashed).

All communication channel is encrypted with a symmetric key. Certificates will ensure that the user is talking to the actual server and authentication from the user will guarantee the validity of the identity to the server. All communications go through the server and redirects them to the destination.



Threats

1. Unauthorized requests for another user's location.
2. The victim's cell phone can be stolen.
3. The attacker gets access to the DB.
4. Since the server is not trustworthy, it cannot have access to the contents of the messages redirected.
5. Eavesdropping.
6. Lack of input validation (SQL injection and buffer overflows).
7. Replay attacks.
8. Man-in-the-middle attack where a malicious user can pretend to be the server.

Basic Solution

For the basic solution we aim to ensure **authentication** through registration (user will be asked to provide phone number, email and the password), login (user will input his/her number and the respective password). [Threat 1]

Also implement the local pin in the app (hashed and stored locally. [Threat 2]

The server will add salt (6 digit) to the plain text password, hash it (SHA-256) and compare with the stored result in the database. [Threat 3]

Encrypted messages ~~symmetric~~ asymmetric key is the solution to the **confidentiality** problem. [Threat 4]

Encrypted communication channels with ~~symmetric~~ asymmetric keys. [Threat 5]

Proper input validation will be assured. [Threat 6]

Intermediate Solution

For the intermediate solution we will guarantee **freshness** through nonces (random 6 alphanumeric code) on register and adding the devices. And timestamps on other types of messages. [Threat 7]

Integrity will be solved by adding a hash(SHA-256) of the message to the message before encryption. [Threat 8]

Advanced Solution

Reauthentication on GPS location update requests will be asked (every 24h) so the session has an end. This is controlled through timestamps, on server side. Two factor **authentication** on adding another device will be added in order to reinforce security. [Threat 1 and 2]

Using signed certificates, the user knows he's talking to the real server. [Threat 8]

Results

We implemented the the initial pin to access the application (stored locally with SHA-256). Register and login with the standard input verification, and a minimum eight-digit password that is stored in the database, added salt (SHA1) and hashed with PBKDF2WithHmacSHA512 algorithm.

The Database has three tables: The login - where the numbers and emails are registered and passwords are stored (hashed of course), the login/register code the user gets to verify the number he is on, and the last logon timestamp to determine if it needs reauthentication. The position - where there is coordinates and the timestamp to when they were last updated, to secure freshness. The connected table - that determines which users are connected to whom. With a nonce that is required upon adding a device for tracking.

The connection between client-server is secure with asymmetric encryption.

And the asymmetric key is from a certificate to prove the authentication of the message.

We use the certificates to generate the asymmetric key pair. They were created by openssl and keytool.

Nonces in order to secure the correct register and the 2 factor authentication are generated with a standard SecureRandom function.

The users are always sending their location to the server and timestamping when they do it, so he knows if the location is fresh or not, in case of the other phone dies out of battery or something.

The user getting the information will know the last time it got sent.

We could not implement some features like the watchdog and the automatic reading of "SMS" but the "SMS" will open and the user just needs to copy and paste the code.

Evaluation

Our project is heavily encrypted because since the integrity/confidentiality of the messages are very important. The passwords in the database are secure, because even if someone gets to it, they cannot get it straight away and to crack it, they will take a considerable amount of time. PBKDF2WithHmacSHA512 algorithm is a hash function with stretching since SHA512 can be computed in the billions per minute with specific hardware[1].

The client app is secure from someone that gets your physical phone, with the pin and, in spite of being just 4/5 digits, to bruteforce it, it will take a considerably amount of time since it exponentially grows the delay. However the file that is stored locally is not protected, just the content.

SecureRandom is the library to use in PRNG (pseudo random number generator) in java [2]

Conclusion

After the results of our solution and comparing them to the proposal we initially ought to do, we completed the Basic solution Intermediate and Advanced solution with some minor exceptions: The channel encryption doesn't use a symmetric key, using asymmetric encryption instead, as the key management for multiple symmetric keys was difficult to handle.

All in all, we think we did a good job splitting tasks and implementing the algorithms that provide the main part of the security in the server and database. We could've improve on some technicalities such better user interface and a little bit more paranoid security locally.

Tool References

- [1] "25 GPUS devour password hashes at up to 348 bilions per second", Adrien Kingsley, 5 Dec 2012
- [2] "Salted Password Hashing - Doing it Right"
- MySQL:
  - Connector to java:
    - https://dev.mysql.com
- Certificate generation:
  - OpenSSL:
    - https://www.openssl.org/
  - Keytool:
    - https://docs.oracle.com/javase/6/docs/technotes/tools/solaris/keytool.html