Weekly Blog - 4

Sree Ram Boyapati	a1775690

What have I done this week and related outcomes?

Learnings

In the last week, I have worked mainly on getting the server running and making the appropriate documentation.

In this aspect, my learning was more in the field of software architecture and cloud infrastructure. The best practices have all been adhered to and able to produce documentation that my teammates can replicate.

Terraform

Terraform[1] lets you create your cloud infrastructure as a code. It maintains your infrastructure as an immutable state. You can define your cloud components as resources and terraform calls the cloud API to provision your servers.

Terraform has certain best practices which I followed -

1. Store state in a remote bucket and use a robot account as the primary user for provisioning.
2. Encrypt the bucket in the cloud storage.
3. Rotate the encryption keys automatically.
4. Do not store/hardcode credentials in the repository.

I was able to achieve these outcomes and used a custom encryption that is symmetric and can be rotated automatically every 30 days as a good security measure.

Challenges:

1. Google Cloud requires enabling of many services APIs and the state management destroys the entire array if we add an element as the API index has been changed. I used a 3rd party plugin [2] to manage the server enablers.

[1] "Terraform." https://www.terraform.io/. Accessed 23 Aug. 2020.

[2] "terraform-google-modules/terraform-google-project ... - GitHub." https://github.com/terraform-google-modules/terraform-google-project-factory. Accessed 23 Aug. 2020.

2. Google service account impersonation was tricky and not many resources on the web were straightforward, alternative was to store credentials of the robot account and pass them to the users. I got it by adding permissions to the user to create a temporary token for use.
3. If the google cloud had already created a component, terraform has to import the resource for the state to maintain parity. I had to import a couple of components.

Python/Flask[3] Web Server

In Rouxi Sun et al. 2020[4], Firebase analytics and firebase backend for mobile applications was found to have a lot of trackers which might do fingerprinting. Instead of letting Google control the server runtime, I chose to host the server myself. I could have used the default exposure notification server implementation[5] given by google for GAEN API. However, It takes a lot of money to host and healthcare interaction is being left out of the scope.

In this regard, my learning included -

1. How to host flask applications on google app engine.
2. How to follow the MVC architecture with regards to the database backend as there are no suitable ORMs. The objective was to hide the inner workings and provide access.
3. How to set up logging to store logs in google cloud so we can follow where the application went wrong.

Challenges

1. Lot of iterations to get the application working on google app engine. Instead of waiting a long times using trial-and-error, I started testing using app engine emulator.
2. FireStore provides a single instance for prod/dev/test. I am presently replicating the collections using prefixing the environment. With more budget, We can host nosql databases to work for our purpose.
3. Composition vs Inheritance - I have used composition in views and services to access the services and repositories respectively. Figure 1 highlights the general MVC framework used.

[3] "Welcome to Flask — Flask Documentation (1.1.x)." https://flask.palletsprojects.com/. Accessed 23 Aug. 2020.

[4] "Vetting Security and Privacy of Global COVID-19 Contact ...." 19 Jun. 2020, https://arxiv.org/abs/2006.10933. Accessed 23 Aug. 2020.

[5] "google/exposure-notifications-server: Exposure ... - GitHub." https://github.com/google/exposure-notifications-server. Accessed 23 Aug. 2020.

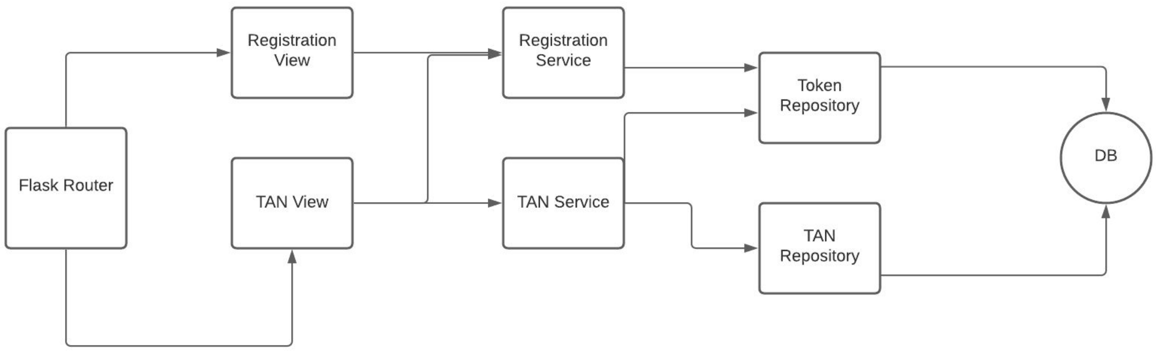


Figure 1: MVC Architecture

Outcomes

1. App Engine is running the code successfully using Flask and new code is going in as versions- https://covidgaurd-285412.ts.r.appspot.com/
2. The entire infrastructure is now in parity with the terraform code. I have written a README.md for my teammates to contribute as well. https://github.cs.adelaide.edu.au/2020-Mobile-and-Wireless-Systems/CovidGuard-F/blob/master/iaac/README.md
3. The server application boiler-plate and for reference, registration of UUID is in progress. Server installation and local testing is also detailed in the README.md https://github.cs.adelaide.edu.au/2020-Mobile-and-Wireless-Systems/CovidGuard-F/blob/master/server/README.md

The API documentation and runner suite is being published using Postman[6].
https://www.getpostman.com/collections/c13298baa16729ab05e7 - Collections of APIs and environments (local/prod) to test the application.

Major Challenges ahead:

1. How do we do rate-limiting without tracking IP address - Anonymous Authentication seems promising. Presently, I plan to expose the API using a weak authentication scheme like BasicAuth for the purpose.
2. Generating batches of exposed infected users keys periodically and storing it in cloud storage and CDN.

With this basic server implementation for demo would be done and We can concentrate on testing the mobile application thoroughly for efficiency, security and privacy.
My paper reviews are based on the efficiency checklist and other parameters that I plan to work with in long term.

[6] "Postman." https://www.postman.com/. Accessed 23 Aug. 2020.

Paper Review # 1

Hatke, Gary F. et al. “Using Bluetooth Low Energy (BLE) Signal Strength Estimation to Facilitate Contact Tracing for COVID-19.” MIT Lincoln Laboratory.

Why have I chosen this paper?
The pose of the user has a role to play in the signal strength for exchanging BLE payloads. This was suggested as an additional aspect to determine the efficiency of the application.

Essentially two parts can come into play -

1. Detecting the location of the user - is the foreground screen dim?
2. Adding offset to proximity estimation based on the location

This paper has done some studies and published some findings that we can use in our approach. Too close for too long (TCFTL) is the metric defined to assess that a close contact has occurred between two users.

Outcome of the Research

1. There are some interesting findings that only 30% of TCFTL cases have been identified by the Apple/Google Exposure detector. According to the paper, theoretically 60% of the cases can be detected by using superior detectors.
2. The paper advises to increase the number of scan periods in the interval of ‘too long’ time for better results, Since BLE is low energy consuming, this finding can be very useful. This is useful to collect more chirps from a person to estimate if they are a close contact.
3. Some details of the GAEN API which we did not know - Chirps produced at 4Hz rate, Scanning is done every 5 minutes with a scan time of 4 seconds and number of scans is 6 in a 15-minute interval. This is important to measure if our implementation is closer to the one defined in the specification.
4. Qualities of a good detector is that it should receive BLE messages for all too-close contacts (within 6m) and so the minimum power should be set where maximum range at which BLE message is possible to detect within 6-8m.
5. After getting the chirps matched, M-of-N matches that they say is close to the contact where M is set threshold should be good enough to confirm the close contact regardless of the position.
6. https://github.com/mitll/MIT-Matrix-Data - Dataset from the research to use as test data for the detector testing.

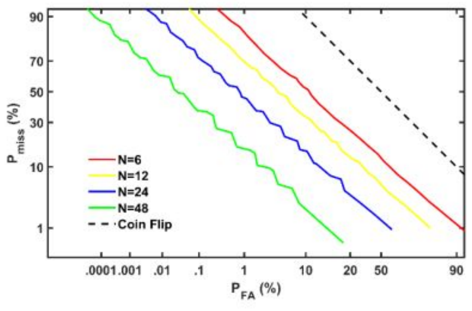


Figure 5: DET curves for M-of-N TCFTL detector for both phones in pants pockets for various values of N

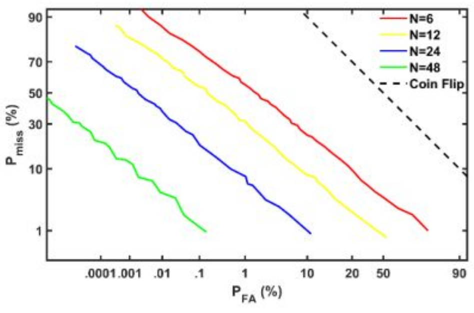
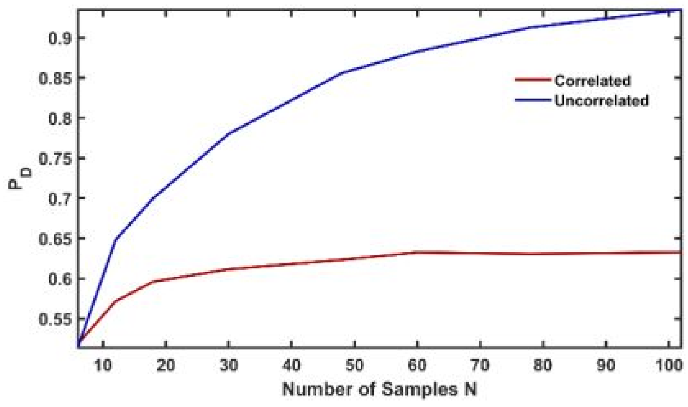


Figure 7: DET curves for M-of-N TCFTL detector for both phones in shirt pockets for various values of N

Performance was measured using Detection Error Curves. P(FA) - Probability of false alarm. P(miss) - Probability of missing the contact (depends on RSSI values).



The probability of detection given a ranges vs Number of samples collected. After 20 samples, Pd increase diminished.

Usage in our project:

Finding from the paper can be useful in many ways in our project

1. Minimum Power level of the BLE scanning to capture all too-close contacts within 6-8m range, This is important for battery efficiency.
2. To take into account the pose of the application, using an M-of-N threshold binary detector is proven to give good performance to detection of confirmed close contacts. We may receive many chirps from the close contact user during a 15 minute interval time.
3. This helped us to understand how the pose of the user might affect if the lower number of samples of chirps are gathered. It will help us set the ideal M value given the N number of chirps collected.
4. If the power of BLE (RSSI) is very high, we may get more false-positives.
5. We shall increase the scan periods to get at least 48 chirps per contact, hopefully the power saved by lowering RSSI can be utilised here.
6. The detector is binary (TCFTL or not) , RSSI and N sample size should determine threshold.

Paper Review # 2

Lindell, Yehuda. "Anonymous authentication." Journal of Privacy and Confidentiality 2.2 (2011).

Why have I chosen this paper?
We can't rate-limit users based on the IP-Address as we would be storing IP/session in that case in ephemeral cache like redis. Also, some form of authentication is useful rather than exposing the API to the public. As of now, I want to restrict the user's access using BasicAuth as we are not building a server for the public to use.

Authentication by registration token has a single point of failure - If the registration token is compromised, the attacker can upload a lot of false-positive chirps.

I would rather prefer some kind of authentication which is anonymous in nature using registration tokens and would like to understand how authentication in such a scenario might work out. Anonymous Authentication by Andrew Y Lindell explores various methods and how anonymous authentication might work.

Also, how can an user be sure that the GAEN-compatible API server cannot ascertain their identity from network traffic. The server might be an honest-but-curious server.

Server might have logs of the client IP-Address from which the payload was sent - revealing the identity of the positive diagnosed user to exposure notification service authorities other than health care providers. The server also knows the IP address of the user as the server provides the registration token. **Anonymous authentication is needed to mask this.**

Outcome of the Research
There are two aspects to achieving anonymous authentication -

1. User Anonymity - Requesting no details from the user and removing any identifiable information that is being collected.
2. Unlinkability - If someone can view internet traffic - They should not be able to know the server they are communicating with.

There are two parts to the protocol -

1. Anonymous Routing
2. Anonymous Authentication on top of Anonymous Routing

A mix server acts as an intermediary to encrypt user identity (for response back) or user message. The 3rd approach blending-in-a-crowd was suggested that the user sends to a random mix server which forwards it to another mix server. Hence, order of the messages sent by the user is preserved if and only if the mix server mesh finally communicates with the destination server in the same order.

Anonymous Authentication is also divided into two parts -

1. Secure Authentication - Other users should not be able to fool the server that they are indeed them. In the registration token approach, if the token is lost, this is possible.
2. Anonymity - The server should not know which user they are interacting with.

In the end, The paper suggests that user uses a method similar to OpenPGP's web-of-trust[7] concept using ring signatures. By authenticating using a ring signature, the server will know it is from a set of valid users and may not be able to find out the person who authenticated it.

How can we use it in our project?

1. Terminology established by the paper can be used to assess our app's authentication security.
2. Introduction to unlinkability as someone who has access to internet traffic (governments) is thought-provoking. With TLS 1.3, Server Name Indicator is being encrypted when packets are in transit. TLS 1.3 support is only added in Android 10 and Android being heavily fragmented, It might take some time to get the update. The approaches suggested can harden TLS 1.2. https://square.github.io/okhttp/ with conscrypt java package can be used to make TLS 1.3 connections in earlier versions.
3. Anonymity of the user is restricted to only tokens that are collected by the user. By assuming the server is honest-but-curious and no other information is provided. However, A routing protocol is needed to mask the IP address for registration and upload of the tokens if the server is not honest-but-curious. https://github.com/guardianproject/NetCipher provides support for onion routing protocol if orbot is installed on android.
4. I don't think ring signatures can be used as people with apps and infected users public keys are not shared or generated in our application. Unless mobile apps create a public key and upload it to the server when they are infected, later other infected users can form a ring signature using these keys - however, how will the zero patient form a ring? threshold size of ring to maintain anonymity? these are not explained.
5. Revocable authentication requires smartcard system which means user's identity is known to a third party. Hence, it may not be possible.
6. We need to use TLS 1.3 for unlinkability by traffic analyser and secure storage of registration token in the android client for secure authentication. Leak of ip-address to server can be prevented by using onion protocol and more research is needed to prevent leak of IP-address.

[7] "Web of trust - Wikipedia." https://en.wikipedia.org/wiki/Web_of_trust. Accessed 23 Aug. 2020.