Authentication Application Documentation

Overview:

This application launches Services for each modality. Services operate when the android application is in the background and allows the user to use other applications while the Authentication Application is running. These services are responsible for collecting data for their respective modalities and writing each piece of sensor dump in a separate file inside their respective directories. So, each file contains one reading which occurs at each sampling period. There is another separate service which handles the scoring of each modality along with the cumulative score for all modalities. This cumulative score will access the scores for each modality in order to calculate itself.

Sampling:

The modalities are broken up into two separate categories: 1) Sensors for gathering information about the outside and 2) Internal phone information for gathering information about the state of the phone. For the sensors, the sampling rate that is set for them is not necessarily precise and does not occur at the sampling period specified. This is mentioned in this link: [SensorManager](http://developer.android.com/reference/android/hardware/SensorManager.html#registerListener(android.hardware.SensorEventListener, android.hardware.Sensor, int)). Thus, I have added an option to down sample the sensors since they happen much faster than the specified sampling rate. These are in the file Constants.java. The sampling rates for each sensor are also listed in Constants.java and are in the units of microseconds. The sampling is usually done in a runnable so they are found in the runnable java files of the directories.

For Internal phone information, this sampling is much more precise and are reliably sampled at the sampling period. Thus, I have not added any down sampling rate. These internal phone information is sampled through the use of timers and so the sampling rates are precisely set. These are listed in Constants.java and are in the units of milliseconds. The sampling is done using timers and so they are done inside the service java files of their directories.

Window Sizes:

Each modalities have window sizes that are associated with each of them. Window sizes specify the number of files (samples) that are kept in the directory for the modality. These files are in turn used to calculate the scores for each modality. These window sizes can be changed in Constants.java

Scoring history:

Each modality has its own history of scores. When a new score is calculated, it is stored into a FIFO queue. The capacity of this queue is declared in Constants.java as well. The queue itself is a class called FifoQueue.java.

Architecture:

The entry point for the application is the SensorDataDumperActivity. This activity is responsible for starting and stopping all services which include the services responsible for sampling modalities as well as the scoring services. It requires that the user enter a username which will be used along with a timestamp as a directory to store all the modalities. This directory is located at: /sdcard/Android/data/com.example.hooligan.accelerometerdatadumper/files/<username>\_<timestamp>. This can be viewed using ES File Explorer which is an application found on the android play store. In this directory contains a directory for each modality with each file inside those directories as one sample reading. Each modality service is responsible for creating its own directory and maintaining its window size and writing to that directory. There is a separate service that is started by the SensorDataDumperActivity, which is responsible for calculating and storing scores for each modality as well as the cumulative score. This service is called the ScoringService.java. Inside this file has timers where each timer is responsible for calculating a score for a modality at a specified scoring period. Inside these functions, the files for each modality are read and used to calculate the modality’s score.

The services for collecting the modality readings are structured roughly the same way except for the Camera modality. Accelerometer, air pressure, ambient light, magnetic, proximity, rotation, and temperature all register with the device’s SensorManager to obtain sampling readings and write them in files that are located in their respective directories. The services associated with battery, call state, camera, connectivity, foreground, location, and screen state use timers to poll the phone’s internal information to gain sample readings. They are found in the TimerTask variables which are located in their respective files. They are all structured the same way, but the Camera is a bit more complicated.

Camera Architecture:

FrontBackCameraService\_2.java is the file that handles the service for capturing front and back images. The timertask is responsible for opening the front camera first and capturing and saving an image and then opening the back camera and capturing and saving the image. The front image and back camera happen sequentially and happen every capturing period. There are several key functions and members that are important to this control flow.

mImageReader is responsible for saving the image once the image has been captured.

mCameraManager is responsible for opening each camera device. Once the opening is successful, it will return to the onOpened() method of mStateCallback. This enables the capturing of the image to be executed and the beginCapture() method is then called.

beginCapture() is responsible for making sure the window size has not been reached and also verifying that the correct filename is associated with the camera device being used (front camera or back camera). It then proceeds to createCaptureRequest().

In createCaptureRequest(), we build the requestBuilder which has options for the capture. The callback is also declared and passed into the method for mCameraCaptureSession.capture(). Once the capture has completed it will return to the callback of captureCallback and the meta data can then be saved about the image.

The images are saved using a class called ImageSaver.java which just writes information into a jpg file.

The meta data is saved using CameraMetaSaver.java and is called when the capture has been completed in the captureCallback. All the meta information is dumped into a file and the type and amount of meta data is dependent upon the device. The nexus 6 includes many fields so that might be a good device to use for obtaining meta information.