UROP Final Presentation – Transcript

SLIDE 1 *(Title Slide)*

Good morning, everyone. My name is Rini Banerjee, and I have just completed my first year of Joint Maths and Computing at Imperial. Today, I will be talking to you about how we can monitor smartphone users' security behaviour, which has been the topic of my undergraduate research placement this summer.

SLIDE 2 *(Motivation)*

So, why conduct any research on smartphone users' security behaviour? Well, previous research done in collaboration with the University of Illinois at Urbana-Champaign suggests that there is a correlation between certain smartphone security behaviours and mental health issues. A questionnaire made to specifically measure **smartphone** security behaviours was used in a user study conducted by the University, and this resulted in the described correlation.

Developing software to help strengthen the claim set out by the University of Illinois at Urbana-Champaign could be beneficial for employers, since they could use this technology to make sure their employees' work phones, which may contain sensitive information, are being kept secure. If employers have information on their employees' mental health conditions, this research could help employers to understand and predict which employees are likely to use the security features on their smartphones in specific ways, which could improve the way employees keep their company's sensitive information private.

SLIDE 3 (*Related Work*)

Another reason for pursuing this research topic is that the work of others in the field has led us to this point. For example, Egelman et al developed a scale which measures general security behaviours, based on a questionnaire which focuses on 4 broad security topics: **Awareness, Passwords, Updating & Securement**. They also validated the claim that self-reported behaviours can actually predict real behaviours.

On the mental health end of the correlation, work by various researchers over the last 10-15 years has examined the link between smartphone behaviours and particular mental illnesses. Some of these correlations include the link between smartphone usage behaviours and bipolar disorder (Alvarez-Lorano), as well as tracking internet usage as a way of passively monitoring depression (Kotikalapudi).

SLIDE 4 *(Problem)*

My task was to develop an Android application, which could be used in the real world to track the behaviours described in the Urbana-Champaign research. Using this app for field studies would improve the accuracy of the correlation between smartphone security behaviours and

mental health, since field studies typically provide more accurate, real-world data points than user studies. The latter tends to suffer from issues, such as the potentially large difference between self-reported behaviours and actual behaviours; conducting a field study would provide more objective evidence to strengthen the research claims.

SLIDE 5

The security behaviours in question can be divided into two groups: **technical** configuration actions, and **social** configuration actions.

SLIDE 6

Technical configuration actions included:

- How the user manages their ad settings (including personalised ads)
- Changes in Bluetooth and WiFi
- How often the user changes their password
- Whether the user uses adblocking and antivirus apps
- Whether the user covers their phone screen when out in public

SLIDE 7

Meanwhile, **social** configuration actions included the following:

- How often the user uses apps requiring sensitive information (such as banking apps, for example)
- Whether the user checks important information about apps they download – for example, developer name
- How the user deals with suspicious texts and emails

SLIDE 8 *(My Approach)*

My approach to this problem involved using different Android components to track relevant security behaviours, including the ones I mentioned earlier. I was given a set of milestones to complete, which were as follows:

<read off slides 9 – 12>

SLIDE 13 *(System Diagram)*

Over the course of my project, I came to realise that Android apps have a very specific structure, which developers must understand and follow. Android does not have a single entry point for execution – like, for example, a main function. Instead, developers have to design applications in terms of components. The most important components for my project were:

- **<click>** First, we have **Activities.** These define the user interface, and typically, there is one activity per "screen" of the app. So, the Main Activity for this app is the one relating to the main screen of the app, which is the first screen you see when you open the app.
- **<click>** Next, we have **Broadcast Receivers**. These subscribe to system events and can initiate different actions depending on what message has been sent by the system. For example, we could make a Broadcast Receiver that checks for the WiFi being switched on and off, so that a message is displayed every time the WiFi state changes. **<click>** Broadcast Receivers listen for messages from the system to inform them of any changes to the variable being tracked, **<click>** and also send messages to the Main Activity, specifying which tracking behaviours have changed so that this information can be recorded.
- **<click>** Now, we have **Handlers**. A Handler allows the same activity to be called several times, which is useful for background processing when there are no inbuilt Broadcast Receivers to do the job for us – for example, we could start a Handler which checks every 5 seconds whether a certain app is running in the foreground. **<click>** Although Handlers do not receive messages from the system automatically like Broadcast Receivers, the Main Activity can save the last known state of the variable being tracked (such as, for example, the device's Advertising ID) and then pass it into the Handler (which takes a Java Runnable object and creates a new thread). The Handler can check whether this variable has changed by **<click>** polling the system at regular time intervals and **<click>** comparing the new value with the stored one. If anything has changed, **<click>** the Handler simply sends a message back to the Main Activity so that this information can be recorded.
- **<click>** Next, we have **Listeners**. These are similar to Broadcast Receivers in that they can listen to events, but Listeners are usually used to detect changes that the user has directly invoked – like, for example, clicking on a button in the app. In this case, an onClickListener could be implemented to execute a certain function whenever that button is clicked. **<click>** Listeners rely on user input, and in this app, **<click>** they relay information back to Main Activity, just like Broadcast Receivers and Handlers.

Activities, Broadcast Receivers, Handlers and Listeners are the main Android components I relied on to make this app. As I mentioned, the Main Activity's primary purpose is to provide a user interface for the app. In this app, the Main Activity **<click>** takes input from the user on what behaviours they want the app to track, and then **<click>** starts the relevant Broadcast Receivers, Handlers and Listeners needed to track these particular behaviours. The red arrows here indicate indirect communication from the user to Broadcast Receivers, Handlers and Listeners, since the user needs to go "through" the Main Activity to start these tracking mechanisms.

SLIDE 14 *(Main Activity UI)*

Just to clarify that last point, the user interface for this app's Main Activity looks like this. The user ticks checkboxes in the Main Activity to specify which behaviours they want the app to track.

In general, every time a security behaviour that is being tracked changes, a message is written to a local file in the app, which records the timestamp, a unique id depending on the type of behaviour and a brief description of what has occurred.

SLIDE 15 *(Broadcast Receivers)*

So, now I'm going to go into a bit more detail about the aforementioned Android components that this app uses to track security behaviours.

I used Broadcast Receivers frequently throughout my app.

SLIDE 16

Firstly, I used a Broadcast Receiver for Problem 1.2, which required the app to track changes in the **<click>** configuration of the "hide device" functionality in Bluetooth settings. However, in the newest versions of Android, I found that the device is always discoverable to other devices when connected to Bluetooth, so I adapted the milestone to just check whenever Bluetooth was switched on or off. For this, I extended the BroadcastReceiver class to make a new class called BluetoothBroadcastReceiver. This checks for the action **<click>** BluetoothAdapter.ACTION_STATE_CHANGED. The **<click>** following 4 states are monitored by the system, and so an appropriate message is written to the local tracking file from this receiver depending on which state the device is currently in.

SLIDE 17

Next, I made another receiver for Problem 1.3, which required the app to **<click>** track changes in the device password. For this problem, I extended the class **<click>** DeviceAdminReceiver (which is itself a subclass of BroadcastReceiver), and simply **<click>** overrided the method onPasswordChanged() to write to the tracking file whenever the phone's password is changed.

SLIDE 18

I also made a receiver for Problem 1.4, which required the app to detect if the user physically covers their phone screen when in public spaces. For this problem, I decided to set up **<click>** geofences, which are circular areas surrounding the specific latitude and longitude of a location of interest. In my app, I used these to set up "trusted places" of 100m radius that the user can input into the app, so that whenever the smartphone is outside one of these trusted places, they are presumed to be in a public place.

I extended the BroadcastReceiver class to make a new class called GeofenceBroadcastReceiver. This checks for a GeofencingEvent, and gets the transition type if there is one. It then **<click>** checks whether the geofence transition was GEOFENCE_TRANSITION_DWELL (meaning the user has been inside the radius of one of their trusted places for some time) or GEOFENCE_TRANSITION_EXIT (which means the user has left

a trusted place). In the case of the latter, it now presumes the user is in a public place and checks for the other conditions of the problem – i.e., whether the phone screen is covered and whether any apps are running in the foreground.

SLIDE 19

Next, Problem 1.8 requires that the app detects **<click>** if the user turns off WiFi when not actively being used. To check whether the internet is in use, I checked if the list of TCP and TCP6 connections on the device was empty. Then, to reduce the number of times the app checked for this behaviour, I set up a WifiBroadcastReceiver to extend the BroadcastReceiver class. This checks for the action **<click>** WifiManager.WIFI_STATE_CHANGED_ACTION, and only if the state of the WiFi is **<click>** WifiManager.WIFI_STATE_DISABLING (i.e., the WiFi is in the process of switching off) does the app read the list of TCP and TCP6 connections.

SLIDE 20

Finally, Problems 2.2 and 2.3 require that my app determines whether the user checks **<click>** information about newly downloaded apps, such as the developer name and the name of the store it was downloaded from. I created an AppInstallBroadcastReceiver, which extends the BroadcastReceiver class, and this checks for the action **<click>** Intent.ACTION_PACKAGE_ADDED. If a new app has indeed been downloaded, my app will then create a pop-up "questionnaire" to test whether the user checked information about this app before or during its download.

SLIDE 21 *(Handlers)*

Next, I'm going to talk about the tracking functions that rely on the Handler component. There is one Handler in my app, which deals with all of the Handler-based tracking, and it polls the system every minute after it has been started.

SLIDE 22

<click> Two of the milestones involve ads.

Problem 1.1 requires the app to track changes in the device's Advertising ID, which is a unique, user-resettable ID for advertising provided by Google Play Services. My app uses an **<click>** AdvertisingClient, as well as the methods getAdvertisingIdInfo() and getId() to get the device's Advertising ID. It then writes the "original" Advertising ID to the tracking file, saves this into a variable, and when it next runs 1 minute later, it checks whether the new value is different from the saved one, and writes a message to the tracking file if this is the case (and so on and so forth).

Another ad-related milestone I needed to complete was Problem 1.5, which requires the app to detect if the user uses adblocking apps. For this problem, I made a **<click>** whitelist of the

top 15 adblocking apps on the Google Play store. When the Handler executes at one-minute intervals, it checks whether the app currently running in the foreground is one of the apps in the whitelist, and if this is the case, a message is written to the tracking file to indicate this.

SLIDE 23

Next, **<click>** we have Problem 1.6, which requires the app to detect if the user uses antivirus apps. I implemented this in a very similar way to the previous problem with the adblocking apps – in fact, the only change is that the Handler goes through a **<click>** whitelist of the top 15 antivirus apps from the Google Play Store.

SLIDE 24

The next problem is **<click>** Problem 1.7, which requires my app to detect if the user uses VPN app(s) when connected to a public network. The Handler checks every minute after it is started to see if the device is connected to WiFi at all, using the isWifiEnabled() method from WifiManager. Then, it checks whether this WiFi is a captive network and whether it has a password. A captive network is one that produces a sign-in page for its users every time they connect to the network – for example, The Cloud and O2 WiFi are both captive networks. This is checked by seeing if the network has the **<click>** NetworkCapabilities.NET_CAPABILITY_CAPTIVE_PORTAL capability. If it has no password or it is a captive network, the WiFi is presumed to be a public network, and then the app checks if a VPN is running. If all of these conditions are met, a message is written to the tracking file.

SLIDE 25

The final problem that relies on the Handler is **<click>** Problem 2.1, which requires that the app tracks the source of the app when the user performs financial and shopping tasks. I implemented this in a similar way to the adblocking and antivirus app problems – I checked whether the current foreground app was part of **<click>** FinanceAppWhitelist or **<click>** ShoppingAppWhitelist. However, in this case, instead of manually adding the first 15 finance and shopping apps I could find on the Google Play Store to my whitelists, I was able to use a Google Play crawler created by George, which provided me with all of the finance and shopping apps on the Google Play Store, organised by number of reviews. **<click>**

SLIDE 26 *(Listeners)*

The last Android component I am going to be talking about is Listeners. I used two Listeners in my app, and both of them were for Problem 1.4, which requires the app to detect if the user physically covers their smartphone's screen when in public spaces.

SLIDE 27

The first Listener I used detected changes in **<click>** how close something was to the proximity sensor at the front of the phone. For this, I used a **<click>** SensorEventListener, and I overrided

the **<click>** onSensorChanged() method to write a message to the file whenever the phone was being covered (i.e., whenever the proximity sensor detected that something was suddenly close to it).

SLIDE 28

The second Listener I used was **<click>** in conjunction with the Google Maps API that I used in this app. In my app, the Google Maps API provides an interface just like the standard Google Maps app, but instead of giving directions to and from places, my app allows the user to either set their current location as a trusted place, or put a marker on the map and set that place as a trusted place (which ties in with what I was saying about trusted places earlier, and the GeofenceBroadcastReceiver).

In order to detect when the map is being clicked on, I **<click>** implemented a GoogleMap.OnMapClickListener and **<click>** overrided the onMapClick() method to place a marker on the location specified by the user when they click on the map.

SLIDE 29 *(Evaluation)*

I have tested all of the technical configuration problems, and my app is successfully tracking almost all of these behaviours. The only problem I am still working on from this set is the GeofenceBroadcastReceiver, which I am in the process of debugging.

As for the social configuration problems, I have completed Problem 2.1, where my app tracks the source of the app when the user is using finance and shopping apps. I have also set up the AppInstallBroadcastReceiver which will be very useful for Problems 2.2 and 2.3, and I have a clear plan of what I would like to achieve in the next few weeks with regards to problems 2.4, 2.5 and 2.6.

SLIDE 30 *(Conclusion)*

In conclusion, this app has been created in order to strengthen the claim that there is a correlation between certain smartphone security behaviours and mental health issues. I hope that its use in a future field study will provide objective evidence that can corroborate this claim, and that this research will lead to improved smartphone security in workplaces, thereby improving the efficiency and security of these workplaces as a whole.

Thank you very much for listening. Are there any questions?

Notes to self

PRINT OUT UROP MILESTONES PAGE TO KEEP AS REFERENCE

MAY NOT INCLUDE THESE TWO SLIDES: