Using Visualizations to Enhance Users' Understanding of App Activities on Android Devices

Chika Eze, Jason R. C. Nurse*, and Jassim Happa
Department of Computer Science, University of Oxford, UK

Abstract

The ever-increasing number of third-party applications developed for Android devices has resulted in a growing interest in the secondary activities that these applications perform and how they affect a user's privacy. Unfortunately, users continue to install these applications without any concrete knowledge of the breadth of these activities; hence, they have little insight into the sensitive information and resources accessed by these applications. In this paper, we explore users' perception and reaction when presented with a visual analysis of Android applications activities and their security implications. This study uses interactive visual schemas to communicate the effect of applications activities in order to support users with more understandable information about the risks they face from such applications. Through findings from a user-based experiment, we demonstrate that when visuals diagrams about application activities are presented to users, they became more aware and sensitive to the privacy-intrusiveness of certain applications. This awareness and sensitivity stems from the fact that some of these applications were accessing a significant number of resources and sensitive information, and transferring data out of the devices, even when they arguably had little reason to do so.

Keywords: Smartphone Security, Privacy, Android, Data Visualization, Visual Analytics, App Permissions, Resources, Information Leakage, Human Aspects, Decision-Making.

1 Introduction

The development and provision of innovative mobile device applications (apps) has fueled the rapid growth of app markets. Android, a key player in this field, currently has over 1,400,000 apps in its official app market – Google Play Store, as at February 2015 [1], which is significantly more than the 19,000 which it had in December 2009. One of the many reasons for this vast expansion of the Android app market is the support for third-party development (via an extensive Application Programming Interface (API)) that gives new apps access to the device hardware, communication channels, user data, and general device settings. These APIs provide developers with access to a multiplicity of sensitive resources such as Camera, Contacts, Location, Microphone, Phone Details (IMEI, IMSI, ICCID), and user information such as the user's current location, contact list, pictures, messages, and so on.

Driven by the desire to explore the latest software and technology, today's mobile users are keen to download and use these third-party apps, unfortunately, with little concrete knowledge of their activities [2]. In an effort to restrict the privileges of third-party app developers, Android has equipped its APIs with a permission-based security system to control each apps' access to private and sensitive device resources as well as users' information. This permission-based system trusts that users have the capability of discerning what permissions should or should not be granted to an app. In the still most used – but now superseded – Android Version 4 and 5 permission models, users were expected to grant all permissions requested by an app before it could be installed on the device. This "all-or-nothing" approach meant that

*Corresponding author: Department of Computer Science, University of Oxford, Wolfson Building, Parks Road, Oxford, OX1 3QD, UK. Tel: +44 (0)1865 273838. Email: jason.nurse@cs.ox.ac.uk.

if a user was not willing to grant all of the requested permissions, then the app could not be installed on the device. Furthermore, there was no mechanism that allowed users to stop an app from using a permitted resource at runtime. This approach has raised a substantial amount of concerns over the years and a significant number of research and publications (e.g., [3, 4, 5]) have comprehensively analysed the model and proposed more efficient ways to approach the problem. Finally, the Android operating system Version 6.0 [6] is now equipped with a new permission model which not only allows users to grant permissions at runtime but also allows apps to inform the users why such permission is needed in the first place [7]. This is a noteworthy improvement to the previous model and it has turned the entire Android security discourse around.

Regardless of whether an app intends to misuse sensitive resources on the device or not, a user still has the right to know how these third-party apps are using the resources. The old Android permission model failed to provide users with an appropriate level of insight into apps' sensitive information processing practices [2]. Hence, users were not often aware that sensitive resources had been accessed, processed, or uploaded by apps. The majority of literature on Android security has focused primarily on the platform and malware rather than users, even though the user's role in the ecosystem is essential as their actions directly affect their security and privacy [8]. The new permission model, on the other hand, informs users when an app intends to access a particular resource but users still do not know what kind of actions were performed on this resource and how it might affect their privacy.

While researching the effectiveness of applications permission request, Benton *et al.* [3] discovered that providing additional text about permission use did not significantly influence the user's installation behaviour. However, adding visual indicators to the provided information achieved a significant effect on the user's installation behaviour for some experimental conditions. Several research articles have also shown that users respond better to visual information, in terms of Android permissions [4, 9] and other contexts [10]; this is especially if that information is appropriately communicated [11]. With this in mind, in this paper we consider the following research questions: If we provide a user with more information using visuals to portray what actions were performed on various sensitive resources at different time intervals by several apps, can the user make better-informed decisions when granting permission requests to apps? Can the user detect an app performing activities that should not have been carried out within a particular timeframe? Can the user detect an app that is performing an activity that does not conform to its functionalities? For example, a flashlight app that sends a user's location to a remote destination.

To assess our approach, we designed and conducted a user study. This involved developing an app called *Appviz* that is able to: continuously collect the data of activities performed by third-party apps based on requested permissions; analyse these data; and create visuals that can portray and explain to users what actions the apps have been performing within specified timeframes. This paper details the results of our controlled user-based experiment. The remainder of this paper is structured as follows: In Section 2, we examine the related work both on communicating Android permission risks to users and communicating through visuals. Section 3 introduces our research goals and details the experimental method that we apply to conduct our investigation. Next, Section 4 presents the findings from the experiment and discusses them in detail. Finally, Section 5 concludes the report and present avenues for future research.

2 Related Work

Former work on collecting and monitoring the activities of apps focuses on forensics research, which is more valuable to corporations, law enforcement and government agencies rather than the users. Lee *et al.* [12] developed a forensic tool which was used in a study to collect data from smartphones at crime scenes. The tool uses an app located on a Secure Digital Card (SD-Card) to extract data from a powered-on smartphone onto the same SD-Card using the Android API. The SD-Card is then processed on a Linux computer before the collected data can be inspected. In an attempt to eliminate the SD-Card requirements and minimize alteration of original evidence source in mobile phones, Yang *et al.* [13] proposed substituting a cloud computing platform for SD-Card. There remains a flaw in both designs as they only collect data instantly and not continuously. As a result, Grover [14] implemented a prototype called DroidWatch, which has the capability of continuously collecting, storing, and transferring forensically valuable Android data to a remote Web server. However, these aforementioned research articles are not focused or beneficial to helping users understand the activities on their devices.

A fundamental prototype system capable of collecting data that can be useful for alerting users to app activities through visualisation was implemented by Isohara *et al.* [15]. The system consists of a log collector in the Linux layer and a log analysis application. The log collector records all system calls and filters events with the target application. The log analyser matches activities with signatures described by regular expressions, based on some selected Linux operating system calls, to detect a malicious activity. According to the researchers, the system was used to evaluate 230 apps and the result shows that the system can effectively detect malicious behaviours on the unknown apps. This implementation was intended as a proposal for a Kernel-based Android Malware Detection and was never considered for helping users understand what the apps are doing

In an effort to help users make more informed decisions during installation time and runtime, previous research articles have concentrated heavily on analysing permissions and trying to communicate them effectively to users. Others have focused more on investigating user's understanding of, and attention to, Android permissions. In a study to evaluate whether Android users pay attention to, understand, and act on permission information during installation, Felt *et al.* [8] discovered that only 17% of the participants paid attention to permissions during installation, and only 3% actually understood all three given permissions. The authors concluded that the current Android permission warning do not help most users make correct security decisions.

Despite improvements in the permission model in more recent android versions, and based on the hypothesis that the Android permission warning is not sufficient enough in helping users make correct security decisions, researchers decided to focus on optimizing the representation of permission to aid users' understanding. Kelley *et al.* [5] designed a short "Privacy Facts" display that alerts the user to the kind of sensitive resource and data requested by an app at installation time. The report from their study recorded that this "Privacy Facts" display led to a decline in the installation rate of apps requesting a high number of permissions. However, participants still reported that they were unsure about the threats arising from apps requesting too many permissions. This motivated Hettig *et al.* [4] in implementing a tool that visualizes risks arising from permissions in the form of worst case examples to demonstrate potential attack scenarios. The report from their study shows that there was a swift decline from 50% to 13.6% in the installation of apps requesting unnecessary permissions. While such a decline rate is ideal, it is important to note that worst case examples are overestimated risk that might lead to a loss of user attention as all app may eventually be considered dangerous [8].

While testing the efficacy of the Android permission-requests system on Amazon's Mechanical Turk, Benton *et al.* [3] discovered that providing an additional text warning about permissions did not affect the users installation behaviours. However, adding visual indicators significantly affected the installation behaviour of users. In an effort to support users with better understandable information, Kraus *et al.* [9] provided graphical displays of statistical information about permissions based on their functionality. The report from their study shows that users install apps requesting lower number of permissions when statistical information is provided together with graphics.

At their core, all of these research contributions are aimed at helping users to make a more informed decision at installation time. However, a significant amount of research has also been conducted, and tools implemented, to analyse Android permissions at runtime. Frameworks such as Saint implemented by Ontang *et al.* [16], Aurasium implemented by Xu *et al.* [17], AppGuard implemented by Backes *et al.* [18], and TaintDroid implemented by Enck *et al.* [19] were all designed in an attempt to enforce runtime policies. However, none of these aim to, or focus on, alerting the user to the activities carried out by apps based on the requested permissions.

Reflecting on the research field, there have not been many detailed studies published on the effect of using visuals to communicate the activities of apps to users. One study worth mention, however, is the visualization tool Papilio, implemented by Hosseinkhani *et al.* [20], that focused solely on visualising permissions of apps. This tool attempts to group apps based on sets of requested permissions, which in turn led to the formation of partially ordered relations; these show the relationships among apps and their requested permissions as well as the reason behind the existence of the partial order. In a study conducted by the authors, they reported that Papilio led them to discover useful security findings about how permissions are being requested by apps. However, they also acknowledged the need to customize their tool for personal mobile devices so as to provide users with visual knowledge of how permissions are being requested by apps.

Takahashi *et al.* [21] introduced the architecture of a system that visualises and issues alerts of security risks to users. The proof-of-concept implementation which demonstrates the architecture's feasibility was documented in their research paper. The system analyses information by monitoring the user's end-to-end communication and providing customizable alerts by directly visualizing risks to the user. This framework is intended to boost users risk awareness and understanding using visualizations.

Even though a significant number of studies have investigated the topic of android permissions and trying to communicate them effectively to users, most users are still oblivious to the actual security and privacy risks posed by the activities of apps. Many other industry and academic works communicate risk (e.g., that an app may have malware) to users using friendly interfaces and visualizations, but they do not consider alerting the users to the ongoing activities of the third-party apps and the risk these apps pose. It is this area that we focus on supporting in the context of this research article.

3 Research Methodology

3.1 Research Aims

The aim of this research study is to investigate the extent to which visualizations can assist a user in making more informed decisions when granting permission requests to apps. This is provided that such visualizations supply the user with information that portrays the activities performed by apps, especially their use of various sensitive resources at different time intervals.

3.2 AppViz Design and Development

Having considered the limitations in effectively communicating Android permission risks to users especially at runtime, we decided to investigate a different communication approach. We designed and developed an app that is capable of continuously collecting data about the activities performed by third-party apps based on requested permissions, analyses these data, and dynamically creates visuals that can explain to users what actions the apps have been performing within specified timeframes. We called this app *Appviz*.

AppViz was implemented to communicate actual apps activities to the user rather than permissions. This was achieved by mapping the requested permissions to the resources that the permission protects and then monitoring and recording the actions performed on such resources. Since studies have shown that users respond better to visual information, AppViz was designed to analyse the activities performed by apps, and then present them to the user in form of visuals to aid understanding and facilitate enhanced decision-making. Our approach is applicable in both Android permission models as a user can specifically tell what an app is doing with a particular resource accessed based on a requested permission.

AppViz was designed to collect two categories of app data: the basic application statistical data that is available through the Android APIs and the log data that is available from the Kernel-Level logs. The system focuses on analysing third-party apps (not native Android apps) as these are considered the most risky for our context. Also, the third-party app has to be in a running state before it can be considered for investigation; if a user never starts an app and the app is not running in the background, then it poses no threat to the user at that point. To collect the Kernel-Level log data required for our analysis, we implemented the approach proposed by Isohara *et al.* [15] i.e. designing a log collector and a log-analyser for AppViz. The basic statistical data monitored and collected by AppViz at a singular point in time via Android APIs is summarized in Table 1 below. For simplicity, these basic data can be primarily classified into CPU, Memory, and Traffic usage. We hypothesise that these data, if presented using appropriate visuals, might help a user know when an app is consuming more resources than is expected at a particular, even when such app is not active.

Table 1: Basic Statistical Data Collected from Android APIs

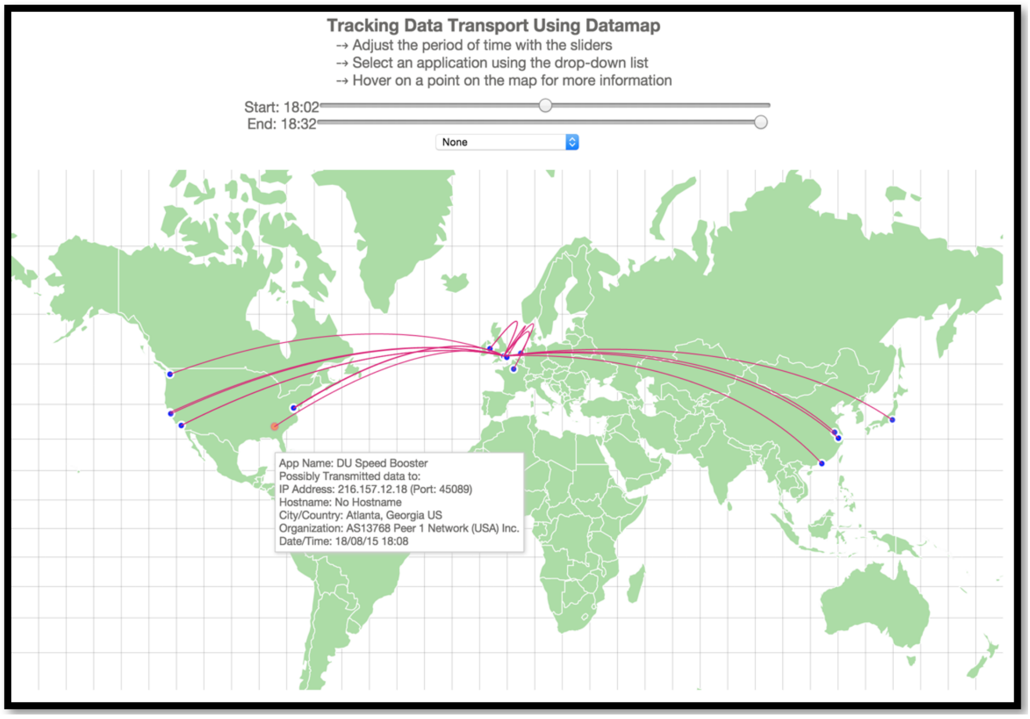
Basic Statistical Data	**Description**
CPU Usage	The amount of CPU used by an app
Total Private Dirty	The amount of private memory used by an app
Total Pss	The amount of shared memory used an app
Rx Bytes	The amount of network bytes received by an app
Tx Bytes	The amount of network bytes transferred by an app

AppViz collects the Kernel-Level log data through the Android debugging tool. This tool is called logcat and it generates log messages that can be viewed using logcat commands via the debugging support tool for Android, Android Debug Bridge (ADB). These log messages comprise of Application-Level logs and Kernel-Level logs for all apps; but for reliability reasons, AppViz uses only the Kernel-Level log messages. To achieve our objective, AppViz analyses the collected app data to determine how to best categorise them for the purpose of creating pragmatic and effective visuals. First it decomposes each data log line into elements using regular expressions. These elements help us to identify individual apps by using their unique process identifier, and then discover the activities they are performing.

Next, AppViz maps these log elements into Events. For the purpose of our research, we define an Event as a 4-tuple: Resource used by an app, Action carried out on such Resource, Details about the action, and Timestamp of when the Action was performed on the Resource. Finally, it categorises these Events based on the apps generating them such that all Events performed by a particular app can be retrieved and treated as a single file. We hypothesise that these Kernel-Level log data, if presented using appropriate visuals, will help users: 1) detect apps performing activities that should not have been carried out within a particular timeframe; 2) detect apps performing an activity that does not conform to their functionalities; and 3) make more informed decisions about what to do with about future request for permissions.

While designing the visualization components of AppViz, we combined ideas from other visualization tools such as MobiViz [22], to display some of the information collected on the Android device in an interactive time chart – this also had the advantage of enabling users sift through the data chronologically. Instead of visualizing permissions of apps like Papilio [20], AppViz maps the permissions into associated resources; and then presents these resources, the actions performed on them, and their timestamps to the users. These visualization components were developed using the Data-Driven Documents (D3) [23] framework because it is flexible, extremely fast and supports large datasets. We also took inspiration from our previous research in terms of system usability, especially as it pertains to security and privacy concerns, and how to communicate risk-related information [24, 11].

AppViz was designed and developed to dynamically create visual diagrams from the analysed data in the Event pool. For our research, four visualizations as shown in figures below, were adopted: Data Map (1), Tree Map (2), Time Chart (3), and Radar Chart (4). The Data Map aims to alert users to apps that are transferring data out of their devices, as well as the locations where these data are being transferred to. The Tree Map informs users on how frequently apps are accessing their resources and the actions that were performed on these resources. With the Tree Map, a user can detect which app used more resources, or which resource was most accessed by different apps. The Time and Radar Charts inform users about the frequency of usage of the device resources by third-party apps within specific timeframes and at a particular timestamp respectively.


Tracking Data Transport Using Datamap
→ Adjust the period of time with the sliders
→ Select an application using the drop-down list
→ Hover on a point on the map for more information

Start: 18:02
End: 18:32
None

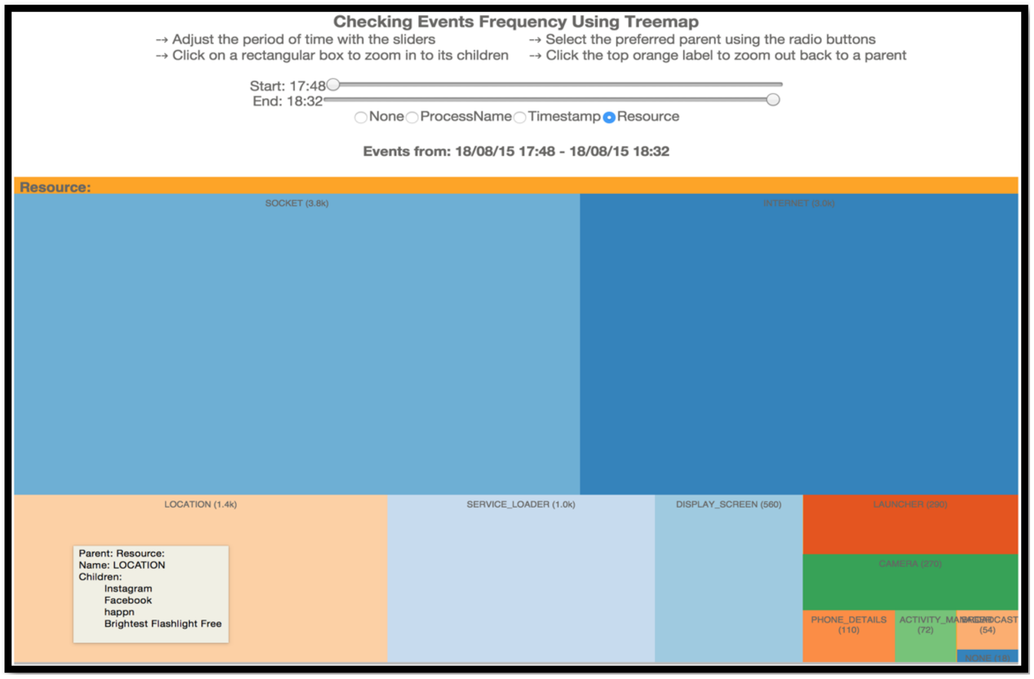
App Name: DU Speed Booster
Possibly Transmitted data to:
IP Address: 216.157.12.18 (Port: 45089)
Hostname: No Hostname
City/Country: Atlanta, Georgia US
Organization: AS13768 Peer 1 Network (USA) Inc.
Date/Time: 18/08/15 18:08

Figure 1: Data Map Visual

In detail, the Data Map Visual in Figure 1 demonstrates that data was transferred to many locations by different apps between 18:02 and 18:32 August 18th 2015. The point currently being hovered over with the cursor (in orange) shows that the app named ‘DU Speed Booster’ transferred data to Atlanta Georgia US on August 18th 2015 at 18:08.

The Tree Map Visual is presented in Figure 2. The visual depicts an interface that has been filtered by ‘Resource’, and therefore shows all the resources that have been accessed by apps between 17:48 and 18:32 August 18th 2015. The number of times that a resource has been accessed is recorded in a bracket right in front of the Resource’s name. For example, Location has been accessed 1400 times. On hovering a mouse on the Location cell, a user can see the parent of Location (i.e. Resource) and the children (i.e. the apps that accessed Location within this timeframe) that would be displayed in another Treemap if the user decides to zoom in. This example says Location was accesses 1400 times by Instagram, Facebook, Happn, and Brightest Flashlight Free.

In Figure 3 the Time Chart Visual is shown and it displays how various apps used the Android device



Checking Events Frequency Using Treemap

→ Adjust the period of time with the sliders
→ Click on a rectangular box to zoom in to its children
→ Select the preferred parent using the radio buttons
→ Click the top orange label to zoom out back to a parent

Start: 17:48
End: 18:32

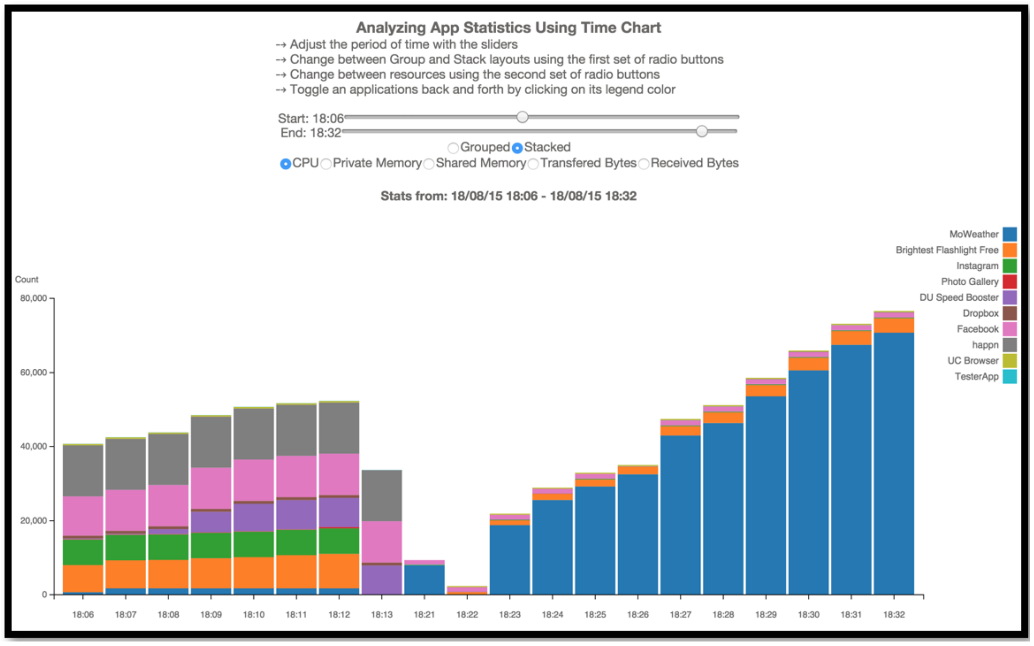
None ProcessName Timestamp [✓ Resource]

Events from: 18/08/15 17:48 - 18/08/15 18:32

Parent: Resource:
Name: LOCATION
Children:
Instagram
Facebook
happn
Brightest Flashlight Free

Figure 2: Tree Map Visual

CPU from 18:06 to 18:32 on August 18th 2015. It shows two sets of radio buttons, one for switching from Grouped to Stacked Bar Chart and vice versa, and the other for allowing the user select what data to compare. The remainder contain the chart itself and the legend.



Analyzing App Statistics Using Time Chart

→ Adjust the period of time with the sliders
→ Change between Group and Stack layouts using the first set of radio buttons
→ Change between resources using the second set of radio buttons
→ Toggle an applications back and forth by clicking on its legend color

Start: 18:06
End: 18:32

Grouped [✓ Stacked]

[✓ CPU] Private Memory Shared Memory Transfered Bytes Received Bytes

Stats from: 18/08/15 18:06 - 18/08/15 18:32

Figure 3: Time Chart Visual

The Radar Chart Visual is presented in Figure 4 and coveys the Memory usage of apps (but CPU and

Bytes usage can also be investigated). A user will notice that this visual has only one slider instead of two like the other visuals. This is because we are only observing the apps' usage as a specific point in time. For example, this Radar Chart tells a user that at 18:31 August 18th 2015, Facebook was using far more Shared Memory than any other app.

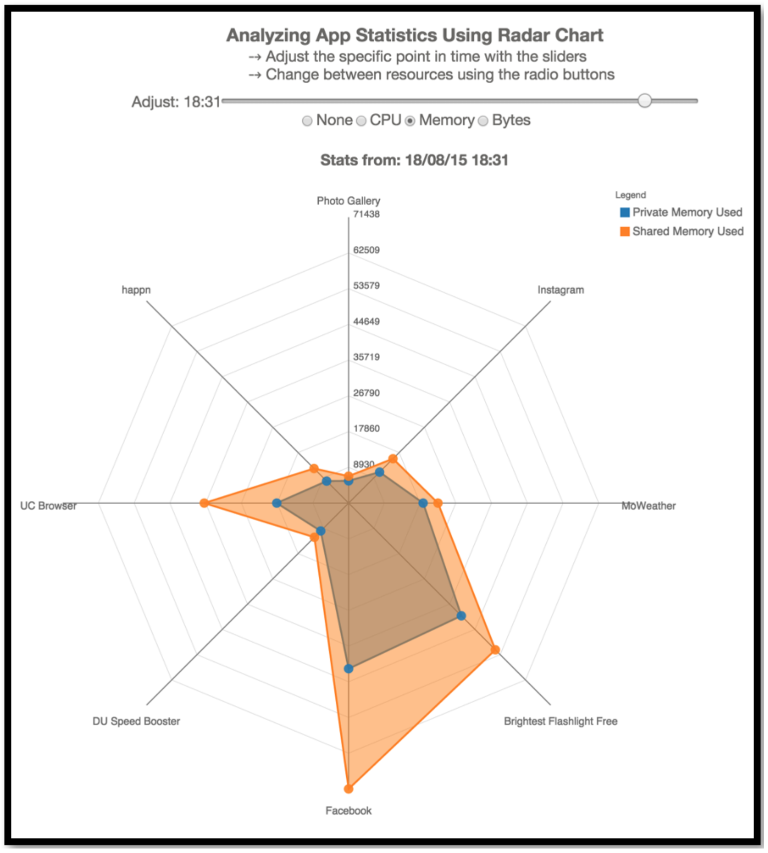


Figure 4: Radar Chart Visual

3.3 Design of the Experiment

To evaluate the capabilities, usefulness, and real-life implementation of AppViz, we planned to conduct a series of user experiments where the system would continuously collect apps' activities data and create visuals for the participants to review. This experiment was to be conducted specifically to evaluate whether users can make more informed decisions based on the information provided by AppViz (and more largely, visualizations).

For this experiment, we used of two Android smartphones running Android version 4.4.2 (Kitkat) and two personal computers with pre-installed Google Chrome for the visualization part of the experiment. Apart from the official Android pre-installed apps, we installed AppViz and 10 other third-party apps on the Android smartphones. These apps were Facebook, Instagram, Happn, MoWeather, DU Speed booster, Brightest Flashlight, Photo Gallery, Dropbox, UC Browser, and TesterApp. These apps were selected because they share the common feature of starting services that constantly run in the background; hence, they may consume a significant amount of CPU, memory and network bytes for their operations. They also have varied access to numerous sensitive device and user resources, which would be useful in attaining answers to our research questions. Some apps were chosen because they perform some specific activities. For example, we decided to use Brightest Flashlight because it transfers data out of the Android device; which of course should not be a permissible function of a typical flashlight. UC Browser constantly transfer data to different locations across the globe even when not in use. We developed TesterApp in-house to access all permissible resources on an Android device; as the name suggests, it was used mainly for testing purposes. These apps were all setup on the smartphones using pre-set data and identities in accordance with ethics and service terms.

The next step involved recruiting participants and collecting data on activities carried out by apps on the Android devices used for the experiment. For recruitment, we chose a convenience sample based on technical background and familiarity (or lack thereof) with the Android platform. We were interested in observing the response of individuals who were familiar with how apps work and those who were not. Given this was a preliminary study, we recruited 10 participants; 3 with a basic background in either cyber security or visualization, while the other 7 with no previous knowledge of either. Also, in the 10 participants, only 5 use or own an Android device as their primary mobile phone/tablet. All participants were students or staff of Oxford University.

Once we recruited the participants, they were briefed in details about the study. We informed them about the flow of the experiment. That is, that they would be using an Android smartphone with some pre-installed apps for some period of time; that they would be given some simple tasks to perform; that they will be shown 4 visuals trying to explain the Android tasks they just performed; and that they would be expected to give their feedback and opinions on the apps' activities based on these visuals. It is this data in particular that we would seek to use in attempting to answer our research questions.

Table 2: The Pre-determined Android Tasks

Tasks to perform during the experiment
AppViz:
1. Tap the "ON" button to power the app
2. Tap the "Start Trace" button to start operation
Facebook:
1. Check-in and upload location using the button at the bottom of the screen
2. Go to photos ⇒ Tap camera ⇒ Take a picture (automatically uploads)
3. Go to Status ⇒ Write a random comment ⇒ Post it on your wall
4. Go to profile ⇒ Change profile picture (Take a new picture or upload existing one)
5. Try to find friends ⇒ Let Facebook find friends for you from your contacts
• Add John Doe as friend ⇒ Cancel Friend Request
Happn:
1. Search for John Doe, then click the love and/or charm button
2. Go to profile ⇒ Add 2 new pictures (Take a new picture and upload an existing one)
3. Go to Invite Friends ⇒ Select Message ⇒ Type 'John Doe' ⇒ Send the message invite
MoWeather:
1. Check the weather report
2. Tap to share report ⇒ Select Messages ⇒ Type 'John Doe' ⇒ Send weather report
DU Speed Booster:
1. Tap the 3-dotted button at the top right-hand side of the screen
2. Select Phone Info ⇒ Scroll down ⇒ Select Hardware Details
Brightest Flashlight: Turn on the flashlight for 30 seconds
Photo Gallery: Search for 'Oxford University' ⇒ Click on any of the results
Dropbox: Click the + sign at the bottom ⇒ Use camera ⇒ Take a picture ⇒ upload
UC Browser: Open www.google.com ⇒ search for 'Oxford University' ⇒ open any result
TesterApp: Click on all the buttons that are displayed to see their intended results.

After the briefing, we presented study participants with a series of structured, pre-determined tasks, and asked them to perform the tasks on the pre-installed Android devices over a specific period of time. The tasks were designed to make apps access user resources and sensitive information. For example, Facebook and Instagram constantly access the Internet as they seek to keep users up-to-date with their current status and feeds. In the tasks, the participants performed operations that allowed these apps to access the camera, location, and contacts. Statistical data of currently running apps was collected every 30 seconds; every 2 minutes AppViz refreshed itself to include any new running apps, log data was collected and processed continuously, and finally all data was uploaded to a secure remote server every 5 minutes. Some of the tasks are shown in the Table 2.

Having the participants perform these tasks largely by themselves was a necessary step before the visualization process to allow participants to become familiar with the operations they perform; this would also help them appreciate the activities that they did not perform in the resulting visuals. We were present during the entire period of the experiment in case the participants had any questions about the tasks or the system interface. Finally, once the participant completed all the tasks, they were asked to close all of the opened apps and to take 10 minutes break; this was done especially to later examine whether participants understood that even after closing apps, the services started by some apps continues to run in background.

The participants were then asked to answer a variety of questions based on the visualization created by the system from the data collected from their previous tasks. These questions were designed based on the tasks performed by the participants and the participants were constantly tested to understand the kind of decisions they would make concerning apps at installation time and at runtime. We were also present during this part of the experiment; however, no help was offered to the participants as we intended them to observe and interpret the visual diagrams based on their own understanding. In Tables 3 and 4, we present a small sample of the types of questions asked to participants – these cover the Data Map and Tree Map Visuals.

Table 3: Questions based on the Data Map Visual

Questionnaire based on Data Map Visual
1. Which country is more data being transferred to by different apps? 2. List all apps that transferred data to that country? *Instruction: Use the drown-drop list to select each app one-by-one while answering the questions.* *Hint: Hover your mouse on the points on the map for more information.* 3. Based on the information you can infer from the map, answer the following questions for all 10 apps: 3.1 Was data transferred out of the smartphone by the app? (Yes/No) 3.2 If data was transferred, to what location(s) i.e. City/Country was it transferred? 3.3 What time did the data transfer occur? 4. Which app transferred more data out of the device and to what locations? 5. Go back through the Tasks you performed on the Android device in Part 1 of this exercise. Can you identify any app(s) that you feel should not have sent out any data from the device at all based on the operations you performed? Why? *Instruction: Adjust the start slider to the time you completed the Part 1 Tasks.* Given that you have closed all the opened apps at that time, please answer the following question. 6. Based on the info you can infer from the map, answer the following questions for all 10 apps: 6.1 Was data transferred out of the smartphone by the app? (Yes/No) 6.2 If data was transferred, to what location(s) i.e. City/Country was it transferred? 6.3 What time did the data transfer occur? 7. Do you think any app(s) is justified to transfer data out of your device even when you have closed such an app? Please explain. 8. If you were wary about your privacy and information leakage from your device, 8.1 Which app(s) would you be hesitant to install in the future? 8.2 Which app(s) would you consider uninstalling from the device right now?

Finally, we collected feedback from the participants by interviewing them to better understand why they made certain decisions in terms of apps they were willing to uninstall immediately and those that they would be hesitant to install in the future. This is discussed in Section 4 below. Each experiment

Table 4: Questions based on the Tree Map Visual

Questionnaire based on Tree Map Visual
Hint: Filter the visual using the provided radio buttons. Hover on an area for more information. 1. Which app is preforming more events on the device? 1.1 App Name: 1.2 What time were more events performed? 1.3 What is the total number of event performed at this time? 1.4 How many resources did the app have access to at this time? 1.5 List the top 3 resources with most number of performed events at this time: *Instruction: Click on the orange labelled bar to zoom out and go back to the top* 2. Which is the most accessed resource? 2.1 How many times was it accessed? 2.2 How many apps were accessing it? 3. How many apps are accessing your location? List them. 4. Go back through the Tasks you performed on the Android device in Part 1 of this exercise. Can you identify any app(s) that you feel should not have access to your location at all based on the operations you performed? Why? *Instruction: Adjust the start slider to the time you completed the Part 1 Tasks.* Given that you have closed all the opened apps at that time, please answer the following question. 5. How many apps are still performing events? List them. 6. Which app is performing more events during this timeframe? 6.1 App Name: 6.2 What time were more events performed? 6.3 What is the total number of event performed at this time? 6.4 How many resources did the app have access to at this time? 6.5 List the top 3 resources with most number of performed events at this time: 7. During this timeframe, which is the most accessed resource? 7.1 How many times was it accessed? 7.2 How many apps were accessing it? 8. Based on your answer in 5, did any (if any) of those app(s) access your location during this time? 9. Do you think any app(s) is justified to continue performing events and accessing sensitive resources even when you have closed such an app? Please explain. 10. If you were wary about private resources being accessed by apps on your device, 10.1 Which app(s) would you be hesitant to install in the future? 10.2 Which app(s) would you consider uninstalling from the device right now?

lasted for approximately 1 hour and 15 minutes. We especially used the experiment to understand the extent to which the visuals influenced their decisions and how any personal bias and preferences affected such decisions despite the information they now possess.

4 Experiment Findings and Discussion

In this section we report on and discuss the results from the experiments conducted. First we present an overview of our observations of the activities performed by the apps. This is followed by a report on the participants' observations about the third-party apps based on the information provided by the visuals. Next, we engage in an in-depth discussion of what these observations highlight, and suggest, based on the feedback and opinions received from the participants.

4.1 Observations about the Third-party Applications

Prior to conducting the study with participants, it was important for us to understand the activities of these apps and how they would perform. Hence, we ran a number of tests to gain some understanding of how the apps were likely to perform when users were presented with the experiment's tasks. With the findings from these tests, we could compare the results obtained from the participants with ours.

Some of our baseline observations include: 1) Facebook and Instagram continually access the Internet, likely because they aim to keep users up-to date with their current status and feeds; 2) Facebook uses more Shared and Private Memory than any other app; 3) Happn, a dating app that relies on a user's location to search for potential matches, transferred more data out of the device than other apps; 4) Brightest Flashlight allows users to use their camera flash as a torchlight, but it also accesses the user's location and transfers that location to an external server – this is not normal functionality for a torchlight; 5) Almost all the apps used in this study run a service in the background even after they have been closed, but this depends on the last operation of the users.

Moreover, 6) Dropbox, an app designed to save files onto a cloud server, used more transferred data bytes than most apps; 7) Instagram and Photo Gallery, constantly download pictures from the Internet, and used more Received Bytes; 8) UC Browser and DU Speed Booster constantly request access to users' phone details and they appear to upload these data to servers in the USA and China even when the device is not in use. With these initial observations in mind, we would expect the participants to be wary about apps such as Happn, DU Speed Booster, and UC Browser when considering the rate of data transfer out of the device; Happn, Facebook, and Instagram when considering the rate of access of user resources such as contacts, pictures, location, etc.; and Dropbox, Facebook, and Instagram when considering the rate of access of the device resources like memory, network bytes and CPU.

4.2 Findings and Results

4.2.1 Data Map

To recap, the Data Map Visual was created to make users aware of apps that are transferring data out of their devices, as well as the locations where these data are being transferred. All 10 participants detected that more data was transferred to the USA and were able to identify the apps sending the data. 7 participants correctly detected that Happn transferred more data. 7 participants affirmed that Brightest Flashlight has no reason to transfer data off of the device. Finally, all 10 participants detected that some apps were still transferring data out of the device even when they had been closed.

To evaluate the effect of the Data Map in helping users make more informed decisions when they are concerned about their privacy and any information leakage, participants were asked to identify which app(s) they would be hesitant to install in the future and those they would consider uninstalling right now. Figure 5 summarizes their responses.

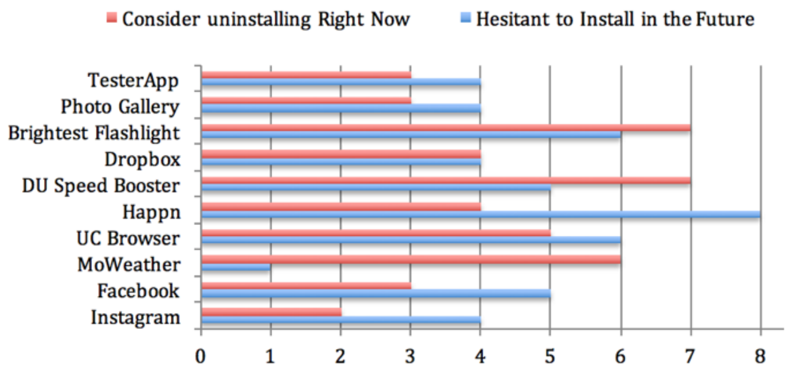


Figure 5: Participants Decision based on Data Map Visual

We observed that more participants were wary about the Brightest Flashlight App. Based on their responses and the fact that 7 participants affirmed that the app should not have transferred data out of the device, it suffices to say that the decision was well informed as expected. Furthermore, we observed that 8 participants (while they do not see an imminent threat of not uninstalling it right now) would be hesitant to install Happn in the future. Since Happn was transferring more data from the device than any other app and was detected by 7 participants, we also considered this to be a well-informed decision.

As expected, the participants were also concerned about DU Speed Booster and UC Browser because they transferred too much data. Interestingly, the participants did not highlight these apps earlier when asked which apps transferred more data around the globe. However, on interviewing 6 participants who made this decision we discovered that this was based on the fact that these apps were transferring too much data to two specific locations – USA and China. Also, 6 participants indicated that they were willing to uninstall MoWeather immediately. We interviewed these participants to understand their decision since MoWeather was not detected to be transmitting more data than is intended for its operation. The response we received from 5 of the 6 participants was that their devices already come with weather apps and that they do not need another. This meant that the decision concerning MoWeather was not entirely based on the visuals, but based on the participants' previous preference. This highlights the natural point that providing visual cues about apps' activities does not lead participants to exclude other decision criteria.

4.2.2 Tree Map

As mentioned earlier, the Tree Map Visual was created to inform users of how frequently apps are accessing their resources and the actions that were performed on these resources. With the Treemap, a user can detect which app used more resources or which resource was most accessed by different apps. In total, 8 participants detected that Instagram and Happn accessed more resources, specifically these 2 apps continually access the Location Resource. All 10 participants correctly reported that the most accessed resource on the device was the Socket, used for connecting to an external network and that Location was constantly accessed by 4 apps. However, 7 participants felt that Brightest Flashlight should not have access to Location given that it is a torchlight. One intriguing finding was that all of the participants noticed that some or all apps were still acquiring resources even after they had been removed from the current list of active Android tasks.

To assess the extent to which the Tree Map Visual may help users in making a more cognisant decision when they are cautious about how private resources are being accessed by apps, the participants were asked to identify which app(s) they would be hesitant to install in the future and those they would consider uninstalling right now. Figure 6 summarizes their responses.

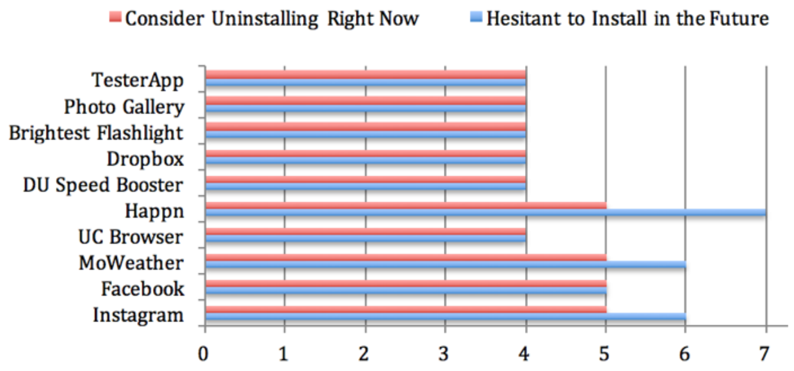


Figure 6: Participants Decision based on Tree Map Visual

We observed that all the participants were concerned about all the apps when considering the way resources were accessed. Based on their responses when interviewed and the fact that 70% of the participants observed that more than 7 apps were still accessing resources even when they were closed, we would say that the decision was well informed. Furthermore, we observed that 7 of the participants would also be hesitant to install Happn in the future (as in the case of the Data Map visual). 6 participants were hesitant to install Instagram and MoWeather in the future, while 5 participants were willing to uninstall Happn, MoWeather, Facebook, and Instagram immediately. Considering the fact that these apps continually accessed location amongst other resources more than every other app, it may be posited that this is also a well-informed decision.

4.2.3 Time Chart

To reiterate, the aim of the Time Chart visualization in our work was to inform users about the frequency of usage of the device resources by third-party apps within specific timeframes. When considering the amount of device resources (CPU, Memory, and network bytes) used by the apps during the entire experiment, 5 participants detected that MoWeather used more CPU, and 9 participants detected that Facebook used more Private Memory. All 10 participants detected that Facebook used more Shared Memory, and that Dropbox used more Transferred Bytes; while 5 participants detected Instagram used more Received bytes compared to other apps. When observing the time period when all apps have been closed and the device was idle, similar results were obtained. Since only 5 out of 10 people correctly detected the behaviour of both MoWeather and Instagram, we hypothesize that the Time Chart visual was not completely effective in communicating the apps activities'. However, more test will have to be conducted before accepting this hypothesis.

To ensure that the Time Chart visual is achieving its objectives in assisting users make well-conceived decisions when they are suspicious about how the amount of device resources used by apps, the participants were asked to identify which app(s) they would be hesitant to install in the future and those they would consider uninstalling immediately. Figure 7 summarizes their responses.

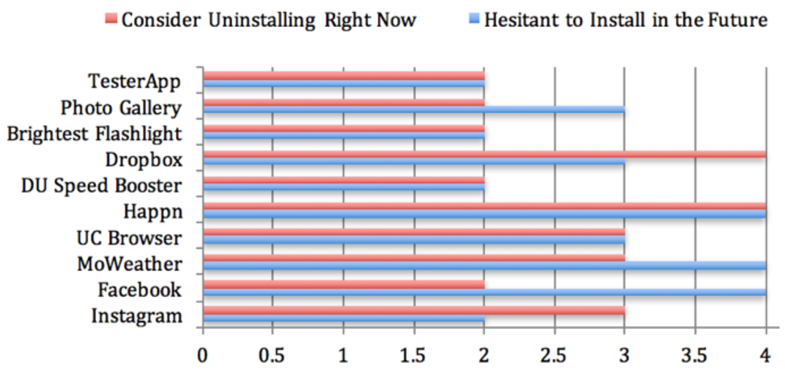


Figure 7: Participants Decision based on Time Chart Visual

We would not entirely regard the participants' decisions based on the Time Chart Visual as appropriate given the context. Firstly, Happn was not using any device resource more than other apps, yet it is the first choice to be terminated by most participants. We hypothesize that it has to do with the fact than Happn did not perform well in the previous visuals presented to the participants; hence, they already formed an opinion regarding the app. However, the decision to rightly uninstall Dropbox and MoWeather is considered well informed as they were using a significant amount of device resources. Due to the reputation of, and services provided by, Facebook, only 4 participants were hesitant to install it in the future and only 2 were willing to uninstall it despite the fact that it used considerably more device resources than any other app. The same can be said for Instagram; only 2 people would be hesitant to install it in the future and 3 will consider uninstalling it immediately despite the amount of device resources used. This result suggests that users might decide to keep using an app depending on their perceived need of it, despite it leaking information. This is a perfect example of the privacy paradox.

4.2.4 Radar Chart

For our study, the Radar Chart Visual was used to inform users about the frequency of usage of the device resources by third-party apps at a particular timestamp. It was intended to validate any conclusions drawn from the Time Chart. Only 7 participants completed this section of the visualization tasks as the other 3 participants felt the tasks were repetitions from the Time Chart Visuals. In total, 4 participants detected that MoWeather used more CPU consistently before and after the apps were closed. 5 participants detected that Facebook used more Shared Memory than Private Memory, but still more than other apps before and after the apps were closed. 4 participants detected that Dropbox used more Transferred bytes than Received bytes, but still more than other apps before and after the apps were closed. This is consistent with the result derived from the Time Chart. Considering space limitations and the fact that the findings from the Radar Chart are very similar to those of the Time Chart, we decided against including a graph here.

4.3 Discussion

Our research goal was to make users aware of the activities that apps are performing on their devices so that they could make more informed decisions when deciding on an app to install in the future or one to uninstall presently. From the demographic questionnaire, we discovered that the 10 participants were often wary about providing personal information to third-party apps. Some of the reasons given were that they do not know how their data is being used; others say that it is annoying and it infringes on their privacy and security. We also discovered that they all desired to know the activities of apps on their devices, which made them invested in the experiment. Participants with technical background provided some justifications on why some apps should be expected to transfer data out of the device, or access both user sensitive and device resources, even after they have been closed. However, the non-technical participants vehemently opposed this view. Their argument was that they expected apps to stop functioning as soon as they were closed. These separate views, nonetheless, did not affect the decisions they made towards hesitating to install an app in the future or considering uninstalling it that moment.

In terms of the main research question of: can users make more informed decisions using these visuals? Our initial results suggest a positive answer – the visuals of the apps' activities affected the decisions made by the users when considering which app to install or uninstall. After the visualization tasks, each participant was asked to pick at most 2 apps that they would be hesitant to install in the future and those they would consider uninstalling immediately based on the entire visuals viewed in the exercise. 8 participants indicated they would be hesitant to install Brightest Flashlight and Happn in the future, and were also willing to uninstall these two apps immediately. We considered this decision to be a reasonable one because Brightest Flashlight was accessing a resource (i.e. location) that does not conform to its functionalities and Happn was transferring potentially too high an amount of data from the device based on its functionality, as well as accessing resources a significant number of times.

The results from our experiment corroborate the findings of previous research which showed that users are more likely to be cautious of installing apps requesting many resources if they (i.e. the users) are presented with visual cues of the apps' intentions. Hettig *et al.* [4] report showed that there was a swift decline in the installation of apps requesting unnecessary permissions when the users were presented with visuals of worst-case scenarios of apps activities. Also, Benton *et al.* [3] discovered that providing users with visual information about app permissions affected their installation behaviours. We achieved the same result by providing the participants with visuals of apps' activities on the Android device used during the experiment.

A significant, but initial contribution of our research is that users can make more informed decisions at runtime using our system. Previous research trying to communicate permission risks to users using visualization are only applicable at installation time; whereas ours can affect the users' decision both at installation time and at runtime. Also, with our system, a user can examine the exact resources accessed, the actions performed on those resources, how often those resources were accessed and actions were performed, and how data is being transmitted out of their own Android devices. This is a functionality that most previous research in this domain lacks, as they were heavily concentrated on analysing and communicating only Android permission risks to the users.

While the sample of participants is small, these preliminary findings still demonstrate the eagerness of users to take swift precaution with their privacy when they are presented with visuals that make the apps' activities more likely to be understood. Further studies have to be carried out on the system with improved data samples as well as more apps to demonstrate how the system would scale. Using custom-built apps might provide a better result as the tasks would be structured to yield expected results based on the design and implementation of these custom-built apps. From the demographic questions, all participants claimed to understand the essence of permissions for apps, and are often wary about providing information to downloaded apps. In further study, it might be useful to consider participants who admit to not knowing the essence of apps' permissions. This way, we can determine if these visuals can trigger or improve user's awareness of apps' permission.

A possible enhancement to the Android operating system is to restrict apps continuous access to sensitive resource until a user's status has changed. For example, if a user has not changed location, then MoWeather should not be allowed to constantly request for the last known location of the user. MoWeather should use the initial data received continuously until it received a notification from Android that the user's location has changed. Another possible enhancement is to integrate a standardized and fully-tested version of our system, or one like it, into the Android operating system. Applications that allow users to monitor how resources are used and how data is transferred out of their devices would be best employed as a system app rather than a third-party app; this is especially with privacy and security in mind. With such a system integrated into Android's primary pre-installed apps, users can constantly monitor the activities of third-party apps. This will invariably help the users in making more informed decisions during installation and runtime of apps.

5 Conclusion and Future Work

The aim of this research paper was to better understand the potential decisions that users make, during the installation of an app and at runtime, based on the visuals of the app's activities. Throughout this paper, we have discussed the concepts, applications, and effects of applying visualizations to enhance Android security, and we have successfully addressed the research questions posed.

To conduct the required research, we developed a system that was able to collect app's statistical data using basic Android APIs and Kernel-Level Log data from the Android Debugging Tool, Logcat. It then analysed the data and classified it according to users-intended display. Four visualizations were created by the system based on the collected app data. Two of the four – Data Map and Tree Map – address how apps access user's sensitive information and resources, and transfer them to remote locations. The other two visualizations – Time Chart and Radar Chart – address how apps use/misuse the Android device resources i.e. CPU, Memory, and Network Bytes. These visuals were able to communicate to the user where data was transfer to by any app, how often was a resource accessed by an app and what kinds of actions were performed on such resources. They are also able to tell the user the amount of device resources an app is consuming which may invariably lead to a declining functionality/speed of the device.

We conducted an experiment where participants were asked to perform a series of pre-defined Android tasks. The data from these tasks were used in creating the aforementioned visuals, and participants were required to use these visuals to understand the behaviour of apps with the aim of making well-informed decisions about the apps used in the experiment. It is evident from the results we obtained that most of the participants were able to detect apps performing activities that should not have been carried out within a particular timeframe, apps performing activities that do not conform to their functionalities, apps transferring data out of their devices, and apps using more device resources than expected. Based on our analysis, we concluded that a user could potentially use the visualization of apps' activities to make well-informed decision regarding installing an app in the future or uninstalling one at runtime.

Further work in this domain should seek to employ even more visualization tools that can help users make better decisions. We would recommend two key visualization tools. Firstly, InstantAlas [25], this tool enables information analysts and researchers to create highly-interactive dynamic and profile reports that combine statistics and map data to improve data visualization, enhance communication, and engage people in more informed decision making. With this tool, not only will the resulting map visual inform users where their data is being transferred to, but will also alert users to the rate at which data is flowing into a particular city/country in comparison to other countries, and how much data is being transferred by an app in comparison to other apps.

Secondly, the TimeLine tool [26] renders an interactive timeline that responds to the user's mouse, making it easy to create advanced timelines that convey a significant amount of information in a compressed space. Each element can be clicked to reveal more in-depth information, thus making it an ideal way to provide a 'big-picture' view, while still allowing access to full detail. This will be a very decent substitute or complement to the Treemap. A user can easily see what resources are accessed by apps and what actions were performed on those resources over time by just moving the mouse to either direction on the visual.

References

[1] Statista, "The Statistics Portal," http://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/, 2015, [Online; Accessed on March 30, 2016].

[2] G. Bal, "Revealing Privacy-Impacting Behavior Patterns of Smartphone Applications," in *Proc. of the 2012 Mobile Security Technologies Workshop (MOST'12), San Francisco, California, USA*, May 2012, pp. 1–4. [Online]. Available: http://mostconf.org/2012/papers/15.pdf

[3] K. Benton, L. J. Camp, and V. Garg, "Studying the effectiveness of android application permissions requests," in *Proc. of the 2013 IEEE International Conference on Pervasive Computing and Communications Workshops (Percom Workshops'13), San Diego, California, USA*. IEEE, March 2013, pp. 291–296.

[4] M. Hettig, E. Kiss, J. Kassel, S. Weber, M. Harbach, and M. Smith, "Visualizing Risk by Example: Demonstrating Threats Arising From Android Apps," in *Proc. of the Symposium on Usable Privacy and Security (SOUPS'13), Newcastle, UK*, July 2013, pp. 1–2. [Online]. Available: https://cups.cs.cmu.edu/soups/2013/risk/paper.pdf

[5] P. G. Kelley, L. F. Cranor, and N. Sadeh, "Privacy as part of the app decision-making process," in *Proc. of the 2013 SIGCHI Conference on Human Factors in Computing Systems (CHI'13), Paris, France*. ACM, April-May 2013, pp. 3393–3402. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2470654.2466466

[6] Google, "Developer Preview 3," http://developer.android.com/preview/overview.html, 2015, [Online; Accessed on March 30, 2016].

[7] Google, “Android Preview: Runtime Permissions,” http://developer.android.com/preview/features/runtime-permissions.html, 2015, [Online; Accessed on March 30, 2016].

[8] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, “Android Permissions: User Attention, Comprehension, and Behavior,” in *Proc. of the 8th Symposium on Usable Privacy and Security (SOUPS’12), Washington, DC, USA*. ACM, July 2012, pp. 3 :1–3:14.

[9] L. Kraus, I. Wechsung, and S. Möller, “Using Statistical Information to Communicate Android Permission Risks to Users,” in *Proc. of the 4th International Worshop on Socio-Technical Aspects in Security and Trust (STAST’14), Vienna, Austria*. IEEE, July 2014, pp. 48–55.

[10] J. R. C. Nurse, I. Agrafiotis, M. Goldsmith, S. Creese, and K. Lamberts, “Two Sides of the Coin: Measuring and Communicating the Trustworthiness of Online Information,” *Journal of Trust Management*, vol. 1, no. 1, pp. 1–20, 2014.

[11] J. R. C. Nurse, S. Creese, M. Goldsmith, and K. Lamberts, “Trustworthy and Effective Communication of Cybersecurity Risks: A Review,” in *Proc. of the 1st International Workshop on Socio-Technical Aspects in Security and Trust (STAST’11), Milan, Italy*. IEEE, September 2011, pp. 60–68.

[12] X. Lee, C. Yang, S. Chen, and J. Wu, “Design and Implementation of Forensic System in Android Smart Phone,” in *Proc. of the 5th Joint Workshop on Information Security (JWIS’10), Guangzhou, China*, August 2009, pp. 1–11. [Online]. Available: http://crypto.nknu.edu.tw/publications/2010JWIS_Android.pdf

[13] C. H. Yang and Y. T. Lai, “Design and implementation of forensic systems for android devices based on cloud computing,” *Applied Mathematics and Information Sciences*, vol. 6, no. 1S, pp. 243S–247S, 2012.

[14] J. Grover, “Android forensics: Automated data collection and reporting from a mobile device,” *Digital Investigation*, vol. 10, pp. S12–S20, 2013. [Online]. Available: http://dx.doi.org/10.1016/j.diin.2013.06.002

[15] T. Isohara, K. Takemori, and A. Kubota, “Kernel-based behavior analysis for android malware detection,” in *Proc. of the 7th International Conference on Computational Intelligence and Security (CIS’11), Hainan, China*. IEEE, December 2011, pp. 1011–1015.

[16] M. Ongtang, S. Mclaughlin, W. Enck, and P. Mcdaniel, “Semantically rich application-centric security in Android,” *Security and Communication Networks*, vol. 5, no. 6, pp. 658–673, 2012.

[17] R. Xu, H. Saïdi, R. Anderson, and H. Saıdi, “Aurasium: Practical Policy Enforcement for Android Applications,” in *Proc. of the 21th USENIX Security Symposium (USENIX Security’12), Bellevue, Washington, USA*. USENIX, 2012, pp. 539–552. [Online]. Available: https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/xu_rubin

[18] M. Backes, S. Gerling, C. Hammer, and M. Maffei, “AppGuard - Real-time policy enforcement for third-party applications,” https://www.infsec.cs.uni-saarland.de/projects/appguard/android_irm.pdf, January 2012, [Online; Accessed on March 30, 2016].

[19] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, “TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones,” in *Proc. of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI’10), Vancouver, British Columbia, Canada*, vol. 49, October 2010. [Online]. Available: https://www.usenix.org/legacy/event/osdi10/tech/full_papers/Enck.pdf

[20] M. Hosseinkhani, P. W. L. Fong, and S. Carpendale, “Papilio : Visualizing Android Application Permissions,” *Computer Graphics Forum*, vol. 33, no. 3, pp. 391–400, 2014.

[21] T. Takahashi, K. Emura, A. Kanaoka, S. Matsuo, and T. Minowa, “Risk Visualization and Alerting System: Architecture and Proof-of-Concept Implementation,” in *Proc. of the 1st International Workshop on Security in Embedded Systems and Smartphones (SESP’13), Hangzhou, China*. ACM, May 2013, pp. 3–10.

[22] Z. Shen and M. Kwan-Liu, “MobiVis: A visualization system for exploring mobile data,” in *Proc. of the 2008 IEEE Pacific Visualisation Symposium (PacificVis’08), Kyoto, Japan*. IEEE, March 2008, pp. 175–182.

[23] M. Bostock, V. Ogievetsky, and J. Heer, “D3 : Data-Driven Documents,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 12, pp. 2301–2309, 2011.

[24] J. R. C. Nurse, S. Creese, M. Goldsmith, and K. Lamberts, “Guidelines for Usable Cybersecurity: Past and Present,” in *Proc. of the 3rd International Workshop on Cyberspace Safety and Security (CSS’11), Milan, Italy*. IEEE, September 2011, pp. 21–26.

[25] GeoWise, “InstantAtlas,” http://www.instantatlas.com, [Online; Accessed on March 30, 2016].

[26] D. F. Huynh, "Timeline," http://www.simile-widgets.org/timeline/, 2009, [Online; Accessed on March 30, 2016].

Author Biography

Chika Eze holds an M.Sc. in Computer Science from the University of Oxford and a B.Sc. from the University of Ibadan, Nigeria. He is an RFID specialist and a patentee for inventing Smart RFID Certificates using Character Proximity. Chika is pursuing various research interests including optimising users' perception of cyber threat, data integrity and identity and access management. He is currently a Cyber Security Consultant with KPMG-UK and has previously worked with the private sector in Nigeria as a digital entrepreneur, and as an ICT Manager at the Nigerian Federal Ministry of Education.

Jason R. C. Nurse is a postdoctoral researcher in the Department of Computer Science at the University of Oxford. He has worked within industry and academia throughout his career. This has included several IT positions within industry, and academic posts such as Research Fellow at Warwick University, and more recently, Researcher at Oxford. Jason has published several articles at both journal and conference levels and also sits on the programme committee of related venues. His research interests include investigating the risks to identity security and privacy online, mobile device security, information security and trust, human factors of security and insider threats.

Jassim Happa is a postdoctoral researcher in the Department of Computer Science at the University of Oxford. His research interests are in visual analytics, cybersecurity, human computer interaction, threat modelling, rendering, and virtual archaeology. He has worked at Oxford since December 2011.