Analyzing Mobile App Privacy Using Computation and Crowdsourcing

Submitted in partial fulfillment of the requirements for

the degree of

Doctor of Philosophy

in

Electrical and Computer Engineering

Shahriyar Amini

B.S., Electrical and Computer Engineering, Cornell University
M.S., Electrical and Computer Engineering, Carnegie Mellon University

Carnegie Mellon University
Pittsburgh, PA

May, 2014

Keywords: mobile privacy, mobile security, app privacy, privacy analysis, crowdsourcing, Android

Abstract

Mobile apps can make use of the rich data and sensors available on smartphones to offer compelling services. However, the use of sensitive resources by apps is not always justified, which has led to new kinds of privacy risks and challenges. While it is possible for app market owners and third-parties to analyze the privacy-related behaviors of apps, present approaches are difficult and tedious.

I present two iterations of the design, implementation, and evaluation of a system, Gort, which enables more efficient app analysis, by reducing the burden of instrumenting apps, making it easier to find potential privacy problems, and presenting sensitive behavior in context. Gort interacts with apps while instrumenting them to detect sensitive information transmissions. It then presents this information along with the associated app context to a crowd of users to obtain their expectations and comfort regarding the privacy implications of using the app. Gort also runs a set of heuristics on the app to flag potential privacy problems. Finally, Gort synthesizes the information obtained through its analysis and presents it in an interactive GUI, built specifically for privacy analysts.

This work offers three distinct new advances over the state of the art. First, Gort uses a set of heuristics, elicited through interviews with 12 experts, to identify potential app privacy problems. Gort heuristics present high-level privacy problems instead of the overwhelming amount of information offered through existing tools. Second, Gort automatically interacts with apps by discovering and interacting with UI elements while instrumenting app behavior. This eliminates the need for analysts to manually interact with apps or to script interactions. Third, Gort uses crowdsourcing in a novel way to determine whether app privacy leaks are legitimate and desirable and raises red flags about potentially suspicious app behavior. While existing tools can detect privacy leaks, they cannot determine whether the privacy leaks are beneficial or desirable to the user. Gort was evaluated through two separate user studies. The experiences from building Gort and the insights from the user studies guide the creation of future systems, especially systems intended for the inspection and analysis of software.

Contents

Acknowledgements

On the surface, a dissertation is the written work of one person. In reality, the work and the writer are influenced by many others, in some cases, years before the dissertation begins. Here, I thank those who have made my dissertation possible. These individuals have influenced me and my work in a positive way. I hope to have the strength and the opportunity to show the same unconditional and selfless kindness, guidance, and support that I have received to others whom I will meet along the way.

I thank my advisor, Prof. Jason Hong, for his generous support, advice, and patience. I thank him for taking me under his wing early on during my graduate career. He has taught me how to ask the right questions, how to present my work and my ideas, and how to persevere. I also thank him for always making time for me and discussing my concerns. He has simultaneously been the toughest critic and the strongest advocate of my work. I appreciate all that he has done.

I thank my other committee members, Prof. Lorrie Cranor, Prof. James Morris, and Prof. Janne Lindqvist for their insightful feedback and instruction. In addition to their support as committee members, each of these individuals has enriched my journey at CMU in other ways. I thank Prof. Cranor for also functioning as one of my qualification committee members, including me in the CyLab Usable Privacy and Security Laboratory (CUPS) meetings, and hosting exciting talks by external guests at CUPS and CMU. I thank Prof. Morris for devoting time to our one-on-one interactions, pointing out fascinating books and lending them to me, and helping me keep the bigger picture in mind. I thank Prof. Lindqvist for his guidance and friendship. It has been a pleasure attending CMU and conferences with him, in addition to the driving range! I have enjoyed many great conversations with Prof. Lindqvist. I also thank him for introducing me to his family and inviting me to his home.

I thank Prof. Priya Narasimhan and Prof. Martin Griss for bringing me to Carnegie Mellon. CMU has been quite a learning experience. This learning experience would not be possible without Prof. Narasimhan and Prof. Griss having faith in me and welcoming me to the university. I also thank Prof. Narasimhan for advising me. I sincerely appreciate her rooting for me and supporting me in my early years as a graduate student.

During my career as a graduate student, I was fortunate to work at several exciting research labs. These experiences helped build the skills and knowhow required to take on a project as large as Gort. My first research internship was at Microsoft Research (MSR), where I worked with Dr. AJ Brush, Dr. John Krumm, Dr. Amy Karlson, Dr. Jaime Teevan, and Dr. Bodhi Priyantha. I thank each of them for their great mentorship. I especially thank AJ Brush and John Krumm. I learned how to conduct my first user study from Dr. Brush. I have been using her teachings to study and evaluate many other systems, including Gort, ever since. I had many enlightening interactions with Dr. Krumm. It was a pleasure to work with him. I aspire to be as well-esteemed and liked by my colleagues as Dr. Krumm is by his.

My internship at Nokia Research Center (NRC) was another great learning experience. I thank Dr. Vidya Setlur for her mentorship, guidance, and support. Even though NRC was going through a lot of changes at the time, Dr. Setlur and the rest of my team at NRC set up an environment where I could conduct useful research. The project was one of the last published works from NRC Palo Alto.

I undertook my most ambitious research work at Google Research under the mentorship of Dr. Yang Li. I consider the project to be my most ambitious one to date based on the amount of effort and work required over the short period of time. Dr. Li frequently asked me great questions and provided excellent insight and guidance. I sincerely thank him for holding me to even higher standards than I did myself. Successfully finishing and publishing my work with Dr. Li reaffirmed to me that I could build very complex systems in a short amount of time and push the boundaries of research beyond concepts and ideas. This was a hallmark experience. I cannot thank Dr. Li enough, for his mentorship brought about the confidence needed to single-handedly build a system as complex as Gort.

The CMU staff have had a key role in enabling this thesis. Their kindness and friendship have been immensely valuable. I especially thank Samantha Stevick, Jennifer Engleson, Indra Szegedy, Samantha Goldstein, Karen Lindenfelser, Susan Farrington, David Casillas, Kevin Topoloski, and JaRon Pitts.

Great companions make for a memorable journey. I thank the members of Chimps Lab for making my journey as a PhD student one that I will cherish for years to come. I thank each of the lab members for their friendship, encouragement, and insightful feedback. I especially thank Afsaneh Doryab for making the lab a friendly and welcoming place. I thank Eiji Hayashi for being my go-to person for HCI methods, sanity checks, and exciting technical discussions. Mike Villena has been a great friend. I thank him for trying out Gort time after time and giving me feedback. I thank Song Luan for being a great friend as well as a great colleague. I appreciate his help with testing and trying out Squiddy. I also thank my friends outside of the Chimps Lab for their kindness and support.

Most of all, I thank my family. They have been the cornerstone of my every success and accomplishment. I thank them for reassuring me that no matter what the endeavor I can rely on their support. I thank my parents for instilling in me the courage to take on any challenge. They have given me the will and the ability to pursue my dreams. I thank my sister, Maral, for being there from the first day that I arrived in Pittsburgh to the day of my thesis defense. I thank her for her kindness, support, and help for as far back as I remember. I also thank her for setting a great example by successfully working on and defending her own dissertation. I also thank Sheila Christian. She has helped me keep a level head, given me great feedback on my work, and helped me present it more clearly.

I thank the funders of this thesis. It would not be possible without them. This research was funded in part by the National Science Foundation under grant CNS-1228813, Google, and a National Defense Science and Engineering Graduate (NDSEG) Fellowship.

1 Introduction

Mobile app stores have responded to consumer demand for mobile apps, with the number of app downloads well into the billions. As of October of 2013, the Apple App Store offers more than one million apps, with over sixty billion downloads [82]. Similarly, Google Play offers Android users over one million apps, with over fifty billion downloads [11]. Considering the availability of context-aware computing resources and users' eagerness to install and use apps, smartphones have become an attractive platform to offer novel apps and services.

Nevertheless, the same appealing features that enable useful services also simplify collecting and inferring of sensitive personal information. For instance, prior to user criticisms, Path, a smart journal app with sharing features, uploaded users' entire contact lists to its servers without user consent [124]. The Federal Trade Commission (FTC) later charged Path with deceiving its users [127]. The FTC has also charged the developer of Brightest Flashlight for deceiving consumers about sharing location information with advertising networks [47]. Unauthorized access to sensitive user data has dire consequences for users as well as service providers. Third party access to sensitive user information may lead to exploitation of users. Examples in the recent past include physical stalking [59] and home invasions [132].

The application market owners are well aware of the high risks associated with installing and running mobile apps, especially malicious ones. Apple, the owner of the largest mobile application market, requires applications to pass through a vetting process before making them available. However, the nature of the process is not disclosed to the public. In fact, there have been malicious applications that had previously passed through the process, before being removed from the App Store in response to user complaints [109]. On the other hand, Google Play relies on the developers to specify the resources used in an application manifest and directly reports this information to the end user [69]. However, it is questionable whether users have the knowhow to clearly comprehend how such resources can be used in malicious ways [54].

Third-party analysts have attempted to address the mobile privacy problem by inspecting apps and bringing privacy issues to the public's attention [4, 78, 81, 124, 126]. However, analyzing app privacy is currently a highly manual and tedious process. A commonly employed approach consists of routing app traffic through an isolated network or a proxy, downloading and testing single apps at a time, interacting with apps manually to inspect their behavior, and making sense of a variety of abstruse outputs [78, 81, 124, 125]. The research community has offered solutions to detect privacy problems using software analysis techniques [40, 43]. However, while these techniques are effective in detecting the use of sensitive information by an app, they do not determine when it is legitimate and desirable.

To further illuminate the problem, consider two applications, a blackjack game and a restaurant finder, both of which request user location, a sensitive piece of user information. Existing techniques would detect that both applications request the user's location, however, the techniques would not be able to determine whether such app conduct is desirable by the user or not. The use of sensitive user information may still be desirable by users given the context of an app. For instance, navigation or trajectory-aware search [7] both require frequent access to accurate user location, which may be still acceptable to users of such applications and services. As such, human intervention, and in some cases, expert knowledge, is necessary to mark the use as legitimate or not.

Prior work has relied on human intervention and expertise to analyze mobile applications. Namely, a Wall Street Journal study of mobile application behavior [126] and also an independent study of mobile application data usage [81] both rely on manual inspection to identify questionable mobile application activity. As experts have to analyze apps one by one, human intervention and the need for expert knowledge create scalability problems. The complexity of determining the legitimacy of sensitive information use combined with the large availability of mobile apps calls for better tools for investigating applications and identifying risky application behavior. To address this problem, this thesis offers the first semi-automated solution to detect the legitimacy of sensitive resource use by mobile apps. To do so, this work tackles the problem through understanding application context by using crowdsourcing.

To determine the legitimate use of sensitive information by applications and also to address the scalability problem, I present a novel approach and a new tool, Gort[1], which rely on crowdsourcing and automation to evaluate mobile application behavior. Specifically, Gort relies on the use of automation and traditional security techniques to learn application behavior in fine granularity. Gort then takes advantage of the wisdom and scale of crowds to evaluate the application behavior. Given the scale of participants available on crowdsourcing platforms such as Amazon Mechanical Turk [6] and the use of automated techniques, Gort effectively enables the analysis and inspection of mobile applications. The approach taken for crowd analysis is briefly discussed in §1.2.

Gort also offers a graphical user interface (GUI) tool to address the needs of mobile privacy analysts. Using the GUI, analysts can examine more applications in detail by taking advantage of automation and also by viewing application behavior in the context of the apps. Moreover, Gort offers a heuristic framework that flags potential app privacy problems. For instance, one heuristic checks if the app has access to both the user's location and the Internet, which would allow the app to track users. The application could send the user's location to a remote server for legitimate or nefarious purposes. Heuristics are further described in §1.1. The GUI tool also reduces the burden of setting up an inspection bench for analysts, which improves scalability in analyzing applications. Refer to Figure 1 for the GUI tool.

[1] Gort is a fictional humanoid robot in the 1951 film, *The Day the Earth Stood Still*.

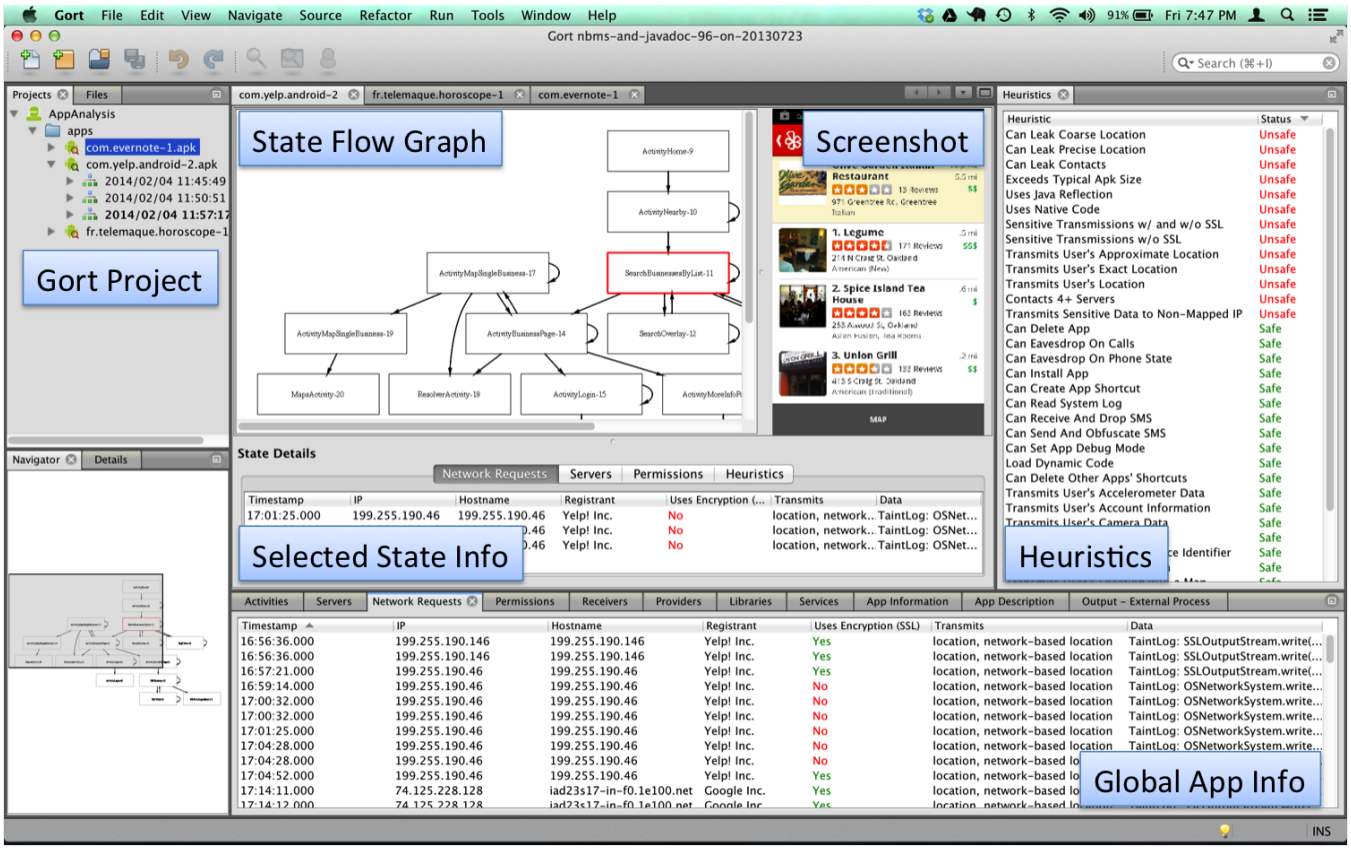


Figure 1: The Gort tool is an interactive GUI for privacy analysts to inspect mobile apps. Using Gort, analysts can investigate detailed app behavior such as the network transmissions sent, the servers contacted, the permissions requested, and also view potential app privacy problems by checking outputs of built-in app privacy heuristics. Analysts can also obtain information from workers on a crowdsourcing platform, namely, Mechanical Turk, about the crowd workers' comforts and expectations about the app's behavior.

Gort is fully extendable to a cloud service that automatically performs static, dynamic, and crowd analysis to mobile applications. In this thesis, I propose a system architecture to bring the Gort analysis engine to the cloud to enable mobile application inspection at scale. This engine will be comprised of all elements of the Gort design minus the GUI tool integrated to perform large scale mobile application analysis. The proposed architecture is the first of its kind to allow mobile app inspection (perhaps software inspection in general), while considering the context and semantics of an application rather than only relying on the application's source code and binaries.

1.1 Heuristics Framework to Flag Potential App Privacy Problems

One of the main research contributions of Gort is a framework for privacy analysts to define heuristics to investigate and evaluate apps. Specifically, heuristics are metrics that analysts can use to find potentially privacy sensitive behaviors of the app. As previously noted, one of Gort's default heuristics is an appli-

cation having access to a user's location and also the Internet. In this case, the app could send the user's location to a remote server for legitimate or nefarious purposes. However, having access to these resources in itself may be a reason for a privacy analyst to further investigate the app. The heuristic framework enables analysts to define their own heuristics and provide code as to how Gort should generate an output for these heuristics. The output for a heuristic is set to be True, False, or Unknown, where Unknown indicates that Gort could not evaluate the value of a heuristic. By providing such a heuristic framework, Gort enables analysts the freedom to choose their own metrics while still benefiting from the automation provided by Gort.

To come up with a set of default heuristics, I interviewed 12 people consisting of 9 security and usability researchers, 2 mobile app developers, and 1 IT professional. During the interviews, I asked the participants to analyze 2 mobile applications to find privacy-intrusive behavior in order to implicitly determine the heuristics they used to examine the apps. After the participants analyzed the apps, I explicitly asked each participant for additional heuristics. This process led to the finding of over 100 heuristics for application privacy analysis. 35 of these heuristics have been implemented in the prototype implementation of Gort and studied during a user study to evaluate the Gort GUI tool.

1.2 Crowd Analysis to Detect Legitimacy of Sensitive App Behavior

Another core new idea to Gort is the use of crowd analysis to analyze app behavior while taking into account the context and semantics of the app. As previously noted, existing tools for app privacy analysis only inform of the existence of sensitive app behavior; however, they cannot determine whether the sensitive behavior was necessary in order to offer users compelling features and services. Gort detects whether sensitive app behavior is legitimate and desirable to users through the use of crowdsourcing. Specifically, Gort uses crowdsourcing to find tasks that can be performed with an app and then asks the crowd questions regarding the resources used by the app while performing these tasks. This approach may be used to evaluate other dimensions of apps (e.g., usability, design), or even extended to software analysis, where analysis requires knowledge of application context and semantics.

Gort uses a three phase crowd analysis approach. First, it presents screens of the app to the crowd and asks for the tasks that can be done using the screens. This phase is called *task extraction*. During the second phase, *task verification*, Gort presents the task labels produced by the crowd in the first phase and asks the crowd to vote on them. Using this process and a voting algorithm, Gort finds the best label for the task associated with the screens. In the final phase of the analysis, Gort asks the crowd whether it makes sense for the application to use the resources that were detected during dynamic analysis of the application while performing the specified task. The final phase is called *resource justification*.

Using the *extraction-verification-justification* paradigm, I performed crowd analysis on 22 apps. From the results, I conclude that crowd analysis is capable of flagging privacy issues with the applications with reasonable accuracy. Such a process can be used in an automated fashion to further extend static and dynamic analysis techniques. The obtained crowd results can be used to study individual applications for example by privacy analysts and developers or by a large application engine to flag suspicious applications. The crowdsourcing approach proposed here is a fundamental contribution to crowdsourcing and mobile application analysis, specifically:

1. The approach uses a non-technical crowd platform to implicitly answer a technical question, which currently requires a great deal of expertise in mobile app development and privacy.
2. The approach opens up a new way to not only evaluate mobile applications with respect to privacy but also to evaluate general applications (regardless of platform) in multiple dimensions (e.g., design, resource usage, privacy).
3. The approach offers the *extraction-verification-justification* paradigm to the crowdsourcing community which may be used for a variety of other crowdsourcing tasks.

1.2.1 Funding Crowd Analysis

Gort uses Mechanical Turk [6], which is a crowdsourcing platform where crowd workers are compensated monetarily. I decided on this incentivization scheme for two reasons: 1. Using a monetary incentive has worked well in the past. 2. Using monetary compensation is a good first step to analyze the system. Other incentive paradigms, such as game-based schemes, could be studied as a part of future work. Nevertheless, paying crowd workers for analysis requires a budget, which begs the question: Who funds the crowd analysis?

A number of feasible solutions may be used to provide the funds for crowd analysis. While numerous such solutions may exist, here, I propose two. One approach would be for market owners to choose a section of their app market for crowd analysis, for instance, the top 1000 apps, and provide the budgets for the selected apps. Using crowd analysis, market maintainers can differentiate the quality of apps by providing a *crowd-approved* badge. Badges are already a common feature on app markets. For instance, the App Store supports the *Editor's Choice* badge [107]. Google Play offers badges for *Editor's Choice* and *Top Developer* [68, 104]. Refer to Figure 2 for an illustration of Google Play's Editor's Choice badge.

Another approach would be for developers of apps to provide the funds for their own app to get a *crowd-approved* badge or certification. In this scenario, at least one crowd analysis engine would have to be run by a market maintainer, an app analytics firm, or a government agency (e.g., Federal Trade Commission). The badge or certification would help developers differentiate their app from other apps that may be more

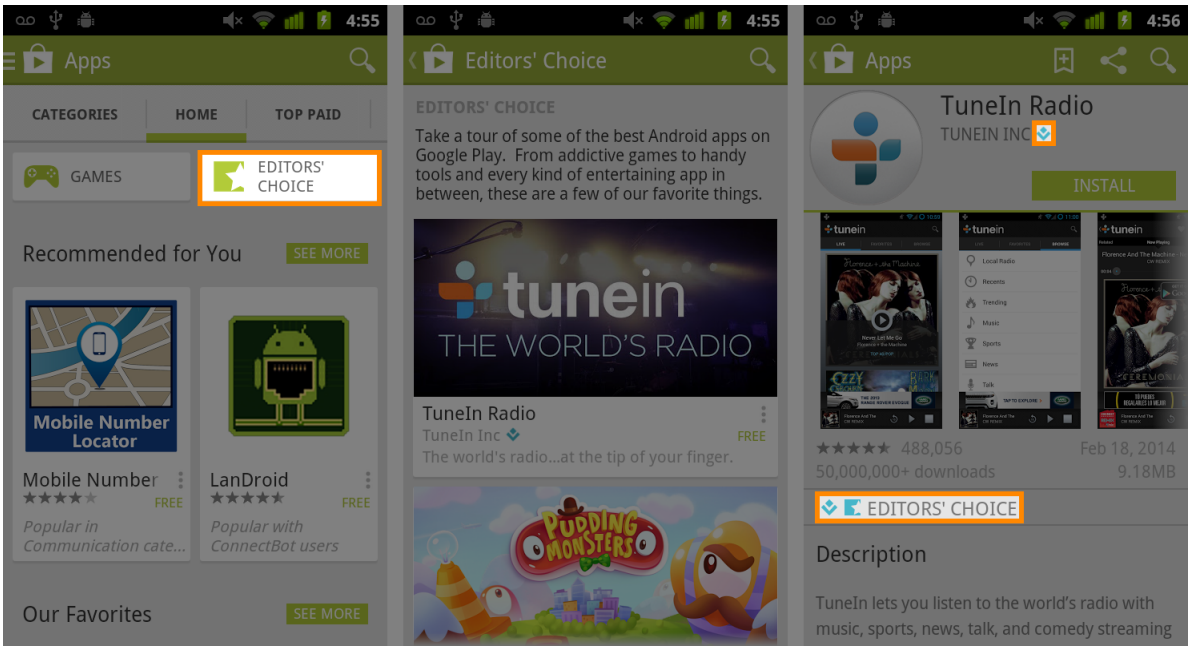


Figure 2: Google Play provides additional visibility to top apps and top developers through the use of badges. This figure illustrates the Editor's Choice and Top Developer badges. In a similar fashion, app markets could feature apps that have a *crowd-approved* badge.

privacy intrusive. Developers already pay fees to join the App Store and Google Play markets and also surrender a percentage of the revenue associated with their app to the market owners [97]. A reasonable fee (e.g., $100) may not be out of the reach for most developers.

1.3 Gort Usage Scenarios

This section presents the usage scenarios of Gort. The mobile application ecosystem consists of many stakeholders including end-users, developers, third-parties (e.g., ad providers, analytics providers, journalists), regulators (e.g., Federal Trade Commission), and app market maintainers (e.g., Google, Apple). The section outlines the usage scenarios for the two components of Gort, the analysis engine and the GUI tool. While the analysis engine may be of more interest to larger entities that would use Gort to analyze a large number of applications, the GUI tool is targeted towards individuals who are interested in inspecting smaller sets of applications or scrutinizing the behavior of a single application at a time.

1.3.1 Gort Analysis Engine for Large Scale App Analysis

Presently both Apple and Google scan applications to detect malicious intent. Namely, Apple runs a vetting process on a candidate application prior to placing it on the App Store. The nature of this process

is unknown to the public. Google scans apps using Bouncer, which provides automated analysis using signature based and behavior based malware detection [95]. While the details of the Apple vetting process are unknown, it would not be unreasonable to assume that both Apple and Google run static and dynamic analysis on apps before and after making apps available on the app markets. The Gort analysis engine supports static, dynamic, and crowd analysis. The static and dynamic offerings of Gort may be integrated into these same processes; however, my speculation is that techniques similar to Gort's static and dynamic analysis are already being used by app market maintainers.

Nevertheless, to the best of my knowledge, at the time of this writing, Gort is the only system to use crowd analysis to analyze applications with respect to the application context. A number of entities may be interested in performing crowd analysis to understand application behavior with respect to application context and semantics. Larger entities could use crowd analysis to filter out applications that are marked suspicious by the crowd or raise red flags that result in sending the application for manual inspection, grading, or approval by an expert. Here I outline a few such entities:

1. Market Maintainers: App market maintainers, such as Google and Apple, are interested in providing high-quality apps to their users and as such already employ application analysis techniques [95]. These entities can use the Gort analysis engine to improve their ability to analyze apps at scale. They may also want to certify applications through crowd analysis to distinguish apps that pass crowd approval from those that do not.

2. Government and Defense Agencies: The National Security Agency (NSA) defined its first phase of Enterprise Mobility Architecture in the Mobility Capability Package in February of 2012. The proposed architecture took advantage of enterprise solutions including the Android platform [101]. Clearly, the use of smartphones is not limited to the consumer markets. Smartphones have long been adopted by government and defense agencies. For these entities, secure use of computing and communication is of utmost concern. As secure mobile computing is mission critical, government and defense agencies would invest in more efficient and thorough scrutiny of the apps that run on their networks. These entities can use the analysis engine to find undesirable or potentially insecure behaviors.

3. Third-Party App Analytics Firms: Market maintainers have not fully addressed the mobile app privacy and security problem. Furthermore, as app market maintainers stand to gain from selling more apps on their market and more ads through their mobile advertisement and analytics arms (e.g., AdMob, iAd), they may not be fully vested in enforcing stricter app privacy requirements. Third-party app analytics firms have emerged to fill the resulting gap for enterprise and consumer

users. A few examples of these firms include Appthority[2], Lookout Security[3], and NQmobile[4]. Similar to the market owners, analytics firms can take advantage of existing static and dynamic approaches; however, they would still not be able to mark resource usage or sensitive transmissions as legitimate without knowing the application context. App analytics firms can use Gort's crowd analysis approach to differentiate their product and also to offer more efficient and thorough app analysis.

1.3.2 GUI Analysis Tool for Small Scale In-depth App Analysis

While the design and implementation of Gort addresses the problem of automated mobile application analysis with respect to application context and semantics, the GUI component of Gort aims to address the needs of privacy analysts. I use the term privacy analyst in a broad sense to refer to the set of users who may have a diverse set of motivations to analyze mobile applications, however, still share the common goal of mobile application analysis. Here I outline the usage scenarios for the GUI component of Gort by highlighting personas interested in using the Gort tool:

1. Information Technicians: Enterprises that embrace *bring your own device* (BYOD) strategy risk sensitive data breaches as a result of employees potentially using privacy intrusive apps on their devices on the enterprise network [23, 50, 128]. For instance, it is not unusual for apps to access users' contacts, e.g., Facebook [73]. Furthermore, apps have uploaded users' entire contact lists to external servers in the past [124]. While uploading users' personal contact list is a problem, such application conduct is much more sensitive in the case of enterprise contacts. Information technology staff at enterprise entities seek to reduce the risk of employees running privacy intrusive or malicious apps. Better mobile application analysis tools would help enterprise information technicians keep suspicious applications from running on their networks. Mobile intrusions are also of extreme sensitivity when it comes to government agencies in addition to enterprise entities.

2. Third-Party Privacy Analysts (e.g., journalists): A number of third-party privacy analysts have exposed suspicious and malicious app behavior to the public [4, 78, 126]. Typically, app inspection requires expertise and involves a tedious and manual process. For instance, in 2010, the Wall Street Journal studied 101 popular smartphone applications for Android and iOS by monitoring network transmissions while manually interacting with the applications [126]. Such approaches do not scale well and act as a barrier to raising public awareness about the potential risks associated with using mobile apps. Better mobile app analysis tools would enable third-party privacy analysts to bring more privacy issues to the public's attention.

[2] `https://www.appthority.com`
[3] `https://www.lookout.com`
[4] `http://www.nq.com`

3. Consumer Advocates: In October 2011, Troy Hunt, a software architect at Microsoft shed some light on the surprising levels of application data rate usage and also data leaks on his own independent blog [81]. In another instance, in February 2012, Arun Thampi discovered that the Path mobile application uploads his contact information to Path's servers while preparing for an unrelated hackathon [124]. This story was later picked up by the media, which helped raise public awareness and resulted in privacy improvements in the Path app [18, 22, 49]. Better mobile privacy analysis tools would enable consumer advocates to analyze applications. This could potentially cause significant change by reducing the technological expertise necessary to analyze apps. Assuming more people could detect suspicious application behavior on their own, more mobile app privacy issues would be brought to light.

4. Public Servants: Following the Path incident, Democratic Representatives Henry Waxman and G. K. Butterfield asked the Apple chief Tim Cook about Apple's iOS app developer policies [3, 18]. Furthermore, the Federal Trade Commission (FTC) charged Path with deceiving its users by collecting personal information from their mobile device address books [127]. Path Social Networking settled the FTC charges [48]. In another instance, the FTC charged the developer of the Brightest Flashlight app for deceiving consumers about how their geolocation information would be shared with advertising networks and other third parties [47]. Better mobile application privacy tools can help public servants and lawyers delve more into the details of mobile application behavior to help them make informed decisions in courts as well as guide privacy policy and regulation.

1.4 Gort Tool Workflow Example

The Gort tool enables privacy analysts to identify potentially risky apps and investigate apps based on fine-grained information generated by Gort regarding app behavior. It is assumed that privacy analysts will start with one or a few apps that they are interested in investigating. A typical workflow using the Gort tool would be as follows:

1. The analyst loads one or a few apps into Gort.

2. The analyst selects from the set of default heuristics already supported by Gort, defines her own heuristics, or uses a combination of the two. An example heuristic would be an app accessing both location and Internet resources on the device, which could potentially be used to track users.

3. Gort automatically scans the loaded apps by performing static analysis and dynamic analysis. Gort also runs the heuristics. Some heuristics can be determined without interacting with the app while others would require Gort to interact with the app by traversing it. The Gort framework covers both.

4. Gort presents the results of its analysis, as well as red flags that may signify potential privacy-intrusive behaviors.
5. The analyst views the details generated by Gort to further investigate sensitive apps, assign a privacy score to the apps, or decide whether an app is approved for usage. If the analyst decides to further investigate the apps, she may also opt to perform crowd analysis on some or all of the apps.
6. If the analyst decided to perform crowd analysis, Gort submits Human Intelligence Tasks (HITs) regarding the apps to a crowdsourcing platform, and presents the combined HIT results to the analyst.

The final output of the Gort tool is the output of the heuristics, the crowd expectations, comfort levels, and associated red flags, and finally, the detailed results of the scans. The privacy analyst decides on how to use this set of information. For instance, one analyst may use a weighted sum of the heuristic outputs to determine a privacy score for apps. While another may use it to approve an app for usage or ban it from use all together, for instance, in cases where security is a critical concern, e.g., enterprise networks.

1.5 Contributions

This thesis makes fundamental contributions to the fields of human computation and also software analysis, especially in the case of mobile applications. Specifically, the work offers a new approach to analyze mobile applications, which focuses on taking into account application context and semantics rather than relying only on application binaries and source code. With respect to human computation, this thesis offers an approach to use a non-technically oriented crowd to answer highly technical questions. Moreover, the proposed Gort system and tool enable detailed traversals of mobile applications and provide privacy analysts with information regarding the privacy implications associated with using mobile apps. Specifically, I am proposing:

> *Through the use of computation and crowdsourcing, Gort offers the first semi-automated approach to analyze mobile applications with respect to context and semantics, which fundamentally changes app (software) analysis to include more information about application behavior rather than just application binary and source code. Furthermore, by providing a system that combines automated traversal of mobile apps, a heuristics framework, and crowdsourcing, Gort enables privacy analysts to investigate apps more efficiently through reducing the burden of instrumenting applications, making it easier to obtain fine-grained privacy information, and presenting sensitive behavior within the context of the application.*

The proposed work will improve upon existing tools for mobile application analysis through four distinct advances over the state of the art:

1. This thesis presents the design, implementation, and evaluation of a heuristics framework for finding app privacy problems. Over 100 heuristics were compiled through studying prior work in mobile privacy, manually inspecting over 40 Android apps for potential privacy problems, and eliciting heuristics from interviews with experts. Out of these heuristics, 35 default heuristics are directly implemented in Gort. Although analysts follow similar principles to inspect application privacy, they do not have an easy way of using cues or red flags developed by other analysts. Analysts can use Gort's heuristics to evaluate mobile app privacy or create their own heuristics by providing code that runs prior to or during app traversal. A heuristics framework with built-in defaults allows analysts to obtain high-level information by taking advantage of common analysis approaches while still allowing analysts to define their own heuristics. The heuristics framework may also be used for large scale app analysis, for instance, to augment analysis flow by filtering out safe apps or flagging suspicious ones. As an artifact of eliciting heuristics by interviewing experts, this thesis also provides an approach to elicit and consolidate heuristics from interviews with experts.
2. This thesis offers two versions of an automated app traversal algorithm, the associated evaluation, lessons learned, challenges, and solutions adopted to overcome these challenges. Gort automatically discovers UI elements in an app's GUI and interacts with these elements to explore various components of the app. As such, the traversal technique does not rely on interactions to be scripted. This approach is more effective than present methodologies which require analysts to manually interact with apps or provide a script to trigger interactions. The proposed solution reduces the cost in time and technical knowhow, while providing fine-grained app behavior. Using the approach, Gort also streamlines the process used to obtain crowd opinion about app context and semantics. The thesis also offers guidelines to improve traversal coverage and overcome challenges that hinder app traversals.
3. The author presents and evaluates a fundamentally new approach to perform software (specifically, apps) analysis by inspecting application behavior while taking into account application context and semantics. This thesis introduces a novel application of crowdsourcing to learn about mobile app context and semantics, and evaluate sensitive app behavior while considering the learned context and semantics. This work also offers a new *extraction-verification-justification* crowdsourcing pattern that can be used for mobile app analysis to account for context and semantics. The crowdsourcing pattern may potentially be applied to other aspects of software, for instance, usability and design.
4. The author has designed, implemented, and evaluated working versions of the Gort system and GUI tool, which is an attestation not only to the feasibility and reliability of the work and ideas presented, but also to their harmonious integration. The author took an iterative approach to building Gort with two separate versions, namely, Gort v1 and v2. Separate user studies were conducted with 12 and 15 users to evaluate the usability of Gort v1 and v2, respectively. Gort v1 required approximately 10,000 source lines of code, written by the author. The latest implementation (v2)

consists of over 35,000 source lines of code by the author, which requires a signification amount of time, effort, dedication, and knowhow. Although the implementation has taken an immense amount of effort by the author, a working system is truly the only way to show the ideas proposed in this thesis really work not only by themselves but also when they are integrated together. To put this effort in perspective, it should be noted that mobile apps exhibit non-deterministic behavior which makes application traversal far from trivial. Furthermore, obtaining reasonable answers to technical questions through crowdsourcing to a non-technical crowd is a challenging problem worthy of its own thesis. In addition to the implementation, the user studies offer a thorough exploration of the system and the tools effectiveness, provide additional insights into the problem, and offer an approach to evaluate privacy analysis tools.

2 Background and State of the Art

This section provides an overview of the mobile app ecosystem and the Android platform followed by a discussion of the state of the art in mobile privacy and security and crowdsourcing as related to Gort.

2.1 Background

Smartphone users obtain mobile apps on app markets. These apps run on a variety of different mobile devices through the use of middleware (e.g., Android, iOS) on top of traditional kernels (e.g., Linux, Darwin) [12, 63]. Mobile apps cater to a diverse set of user needs and offer services across many different domains. To better address user needs, apps have access to a variety of device resources and sensors, some of which provide sensitive information regarding the user and the user context, for instance, GPS location.

2.1.1 Application Markets

Prior to the introduction of Apple's App Store in 2008, users of early smartphone platforms such as Symbian OS, RIM BlackBerry OS, Microsoft Window Mobile, and Palm OS, would find mobile applications by visiting app aggregation or developer websites or would purchase apps distributed on physical media [19]. Once users obtained the application they would then install the software through the use of a host PC and a USB cable.

This model of sales and distribution of mobile applications placed the burden of providing and obtaining mobile apps on developers and users, respectively. Developers had to deal with popularizing, distributing, and selling their applications. On the other hand, early smartphone adopters had to overcome the burden of discovering and installing applications. The Apple App Store was the first application market to reduce this burden on the users and developers by offering a centralized market to purchase and sell mobile applications for the Apple iOS platform. The App Store also popularized the use of the term *App* to refer to a mobile application. Since then, there has been a surge in the use of smartphones and mobile applications. Furthermore, all major smartphone platform maintainers offer their own application markets, e.g., Apple App Store, Google Play Store, and Windows Phone Marketplace. However, other third-party players have also established their own application markets, e.g., Amazon Appstore for Android [5].

App markets offer considerable security utility, as outlined by Enck [42]. First, they allow market maintainers to implement a walled-garden, where the market maintainer has exclusive control over what applications users install. The Apple App Store is an example of one such walled-garden, where iPhone users can only install apps that are available to them through the App Store. Second, app markets act as a

choke point for application security certification. In particular Apple runs a vetting process on a candidate app prior to placing it on the App Store. The nature of this process is unknown. Google scans all apps using Bouncer, which provides automated analysis using signature based and behavior based malware detection [95]. Researchers have bypassed Bouncer through fingerprinting techniques in the past [105]. Finally, app markets can provide remote software management. For instance, Google has used remote software management to remove malicious apps from devices in the past [67].

For the purposes of this work, the focus will be on the Android operating system. This decision was made based on the open source nature of Android as well as the availability of tools and extensions for Android (in particular, TaintDroid [43], and TEMA [122]). As this thesis focuses on the Android platform and Android apps the remainder of this overview information will discuss the Android platform, however, research work related to iOS is still covered.

2.1.2 Android

Android is an open source mobile platform owned by Google [61]. The core phone functionality as well as Android apps run on a customized middleware written in Java and C/C++. Android apps are written in Java and compiled to Dalvik Executable (DEX) byte-code. Applications may include or call native methods for performance optimizations. Apps run on the Dalvik Virtual Machine interpreter[5]. Each app executes on its own Dalvik VM interpreter instance and each VM instance executes as a unique UNIX user. In this way, Android isolates applications from each other. Apps are distributed as Android Package (APK) files, which are Zip archives containing the compiled Java classes in DEX format and other application resources.

Android apps are developed using the Android Software Development Kit (SDK). Apps consist of four main component types: activity, service, content provider, and broadcast receiver [64]. App UIs consist of one or more activities. A service is a component that runs in the background and performs operations (e.g., audio playback). Services do not provide a user interface. Content providers manage app data (e.g., file system, SQLite database). Finally, a broadcast receiver is a component that responds to system-wide broadcast announcements (e.g., device boot complete, SMS received). Three of the components, namely, activities, services, and broadcast receivers are activated by asynchronous messages called intents, which bind individual components to each other at runtime.

Prior to running any app, the Android system must know which components exist in the app. Developers describe the components in their app in an XML file, named AndroidManifest.xml (the "manifest" file), during their development process. Developers also declare any resources that their app requires dur-

[5] A new Android Runtime (ART) virtual machine is being experimentally introduced in Android 4.4 [66].

ing runtime through the use of *Permissions*. For instance, a developer may request permissions to access to the Internet or to read the user's contacts. After a software build of an app, the manifest file containing the app metadata is converted into a binary XML file and included in the app's package.

Permissions: Android allows mobile apps to access device resources by requesting permissions. Android permissions are checked at install-time by requiring users to approve the permissions requested by the app. This is as opposed to the iOS model where access to resources is checked upon time-of-use, e.g., prompting the user to allow an application to use GPS location. Au *et al.* provide a comparison of permission and resource use declaration in different mobile platforms [13]. Prior studies have identified that users pay little attention to permission screens during app installation and generally have a limited understanding of app permissions [54, 87, 134].

2.2 State of the Art

This thesis builds on a number of different topics. As such, this section covers prior work from a number of different areas, namely, work related to privacy and security, crowdsourcing, and user interfaces. This section first discusses approaches related to mobile application protection and analysis. For a comprehensive review of related work to mobile application privacy and security until 2011, refer to an overview of techniques by Enck [42]. The discussion of work in privacy and security here uses the same categorization of past work as provided by Enck, however, only works closely related to Gort or not covered by Enck in 2011 are highlighted. Finally, this section delves into prior work in crowdsourcing and interfaces.

2.2.1 Protection Mechanisms

A number of protection mechanisms beyond centralized market control and permissions have been proposed by researchers. This section discusses the proposed solutions and also their limitations.

Rule-Driven Policy Approaches: Rule-driven policy approaches offer policy languages to analyze mobile applications. Kirin is a security policy extension for Android, proposed by Enck in 2009 [45]. Kirin focuses on dangerous combinations of permissions and action strings to flag applications prior to installation or to display risk statements. Saint, developed by Ongtang *et al.*, is another rule-based approach, which defines install-time and runtime policies placing dependency constraints on permissions and caller and callee constraints at runtime [106]. CRePE is an access control system for Android that enforces policies based on context, such as location, time, and temperature [35]. The Android Permission Extension (Apex) allows users to select which permissions an application is granted [102]. XManDroid addresses permission privilege escalation attacks in Android by preventing confused deputy attacks and

collusion between apps [25]. While rule-driven policy approaches are effective, they have a limitation in that the rules must be effective to capture application behavior and they need to be maintained.

High-Level Policy Approaches: High-level policy approaches focus on overall high-level security goals for smartphones. One example of such a goal is isolation between apps. A practical implementation of isolation is the use of separate Dalvik VM instances for each app in Android. TrustDroid proposes lightweight domain isolation for Android through the use of domains for system, trusted and untrusted domains and using a policy to prevent interaction between trusted and untrusted domains [26]. High-level polices have also been proposed to prevent confused deputy attacks in Android by Felt *et al.* and also Dietz *et al.* [38, 55]. Pearce *et al.* propose privilege separation for apps and advertisers in Android [110]. Finally, Jeon *et al.* advocate finer-grained permissions for Android apps and present a solution, Dr. Android and Mr. Hide, to rewrite Android mobile app bytecode to talk to a privacy library layer instead of the Android APIs [84].

Faking Sensitive Information: Leaking sensitive information has been studied in the domain of traditional computing, and remains to be a problem in the mobile computing domain as well. A number of prior works explore privacy leaks in the non-mobile computing space. Aura *et al.* found that all layers of the protocol stack leak various plaintext identifiers of the user, the computer, and their affiliation to the local link [14]. Privacy Oracle finds potential data leaks using black box differential testing and running an application under different settings [86]. The WiFi Privacy Ticker displays information about sensitive data sent from the computer to the network [34].

Studies have highlighted information leaks in the mobile domain [4, 40, 43, 81, 120, 121, 124, 126]. A number of works present methods to block data leaks to the network or fake the capabilities of the phone or the information provided to third parties [79]. Bereford *et al.* proposed MockDroid to provide fake or "mock" information to apps [20]. TISSA takes a similar approach to MockDroid, however, provides greater flexibility by allowing users to choose from empty, anonymized, or fake information [137]. Hornyack *et al.* present AppFence [79] which substitutes fake data for phone identifier and location, and blocks user specified network transmissions by detecting the transmissions through the use of TaintDroid [43]. ProtectMyPrivacy enables users to provide fake information to apps on jailbroken iPhones [1]. The authors of ProtectMyPrivacy have also developed a crowdsourced recommendation-engine which collects protection decisions by the users and makes privacy recommendations for apps.

2.2.2 Application Analysis

App market maintainers perform analysis on apps. For instance, Google uses Bouncer to analyze apps that are uploaded to Google Play [95]. However, the analysis performed is not always enough and questionable applications still make their way to the markets. Researchers have devised a number of permission,

static, dynamic, and cloud-based analysis solutions to inspect apps. While these approaches are useful in detecting the use of sensitive resources in apps, their key limitation is that they do not identify whether the use of resources or permissions is legitimate or beneficial to the smartphone user.

Permissions: A number of prior works involve the analysis of Android permissions to protect against and detect suspicious apps. Enck *et al.* developed Kirin which identifies when applications use a dangerous combination of permissions and/or action strings. They studied 311 top free apps from Google Play and flagged 10 applications based on rules, 5 of which were still suspicious after reviewing the apps. Barrera *et al.* performed permission analysis on 1,100 apps from Google Play and used Self Organizing Maps to show the frequency of use for combinations of permissions. They found that most apps request a small number of permissions and that there is no correlation between app categories and permission requests. Similarly, work by Felt *et al.* analyzed 100 paid and 856 free apps from the Android Market and found that most apps request a small number of permissions [53]. They found that the 'INTERNET' permission is the most frequent permission requested by apps and that developers make errors in requesting permissions. In follow up work, Felt *et al.* created a tool, Stowaway, to detect over-privileges on applications based on a mapping of Android APIs and permissions [51]. Vidas *et al.* created an Android SDK extension which helps developers find the minimum set of permissions they need to include in their app [129]. Recently, Franke *et al.* used data mining approaches to find request patterns in Android applications. They found over 30 common patterns of permission declarations by apps using matrix factorization approaches [56]. Book *et al.* explored the use of application permissions by ad libraries by inspecting 144,000 mobile applications. They found that ad libraries' use of permissions has increased significantly over the past several years. [24]. Gates *et al.* offer approaches to generate summary risks for apps using permission-based risk signals [58].

Static Analysis: Researchers have explored static analysis methods for both Android and iOS mobile platforms. Static analysis can be done with or without source code, depending on whether the source code is available in addition to the application binary. Egele *et al.* proposed PiOS which performs taint analysis directly on app binaries for iOS [40]. PiOS uses control flow analysis on a CFG generated on the application binary and followed by data flow analysis to confirm leaks. The authors examined 852 free apps on the App Store and 582 apps from Cydia's BigBoss repository. They found that more than half of the apps leaked privacy sensitive device IDs.

Other researchers have explored static analysis approaches using the Android platform. Chin *et al.* explored an approach for Android that analyzes disassembled DEX bytecode [31]. The authors developed a tool ComDroid based on this approach and used it to analyzed 50 popular paid and 50 popular free apps and manually inspected the results of 20. Using their tool they found 34 exploitable vulnerabilities. IPC Inspection [55] and Stowaway [51] are other tools built on ComDroid by the same group. Enck *et al.* proposed a tool, ded, to reverse engineer Android apps to Java. They decompiled and analyzed

1,100 popular apps using the tool [44]. Other researchers have offered source code analysis approaches. Chaudhuri has proposed a formal model for tracking flows between apps using permissions [30]. Fuchs *et al.* created SCanDroid for automated application certification using the WALA Java bytecode analysis framework [57]. Grace *et al.* propose RiskRanker which uses malware samples and their signatures to detect zero-day attacks in a scalable fashion [70]. Gibler *et al.* developed AndroidLeaks and evaluated it using 24,350 Android applications from several markets. AndroidLeaks found 57,299 potential privacy leaks in 7,414 apps out of which 2,342 apps were manually verified to leak private data [60]. MalloDroid is a tool created by Fahl *et al.* that uses static analysis to detect Man-in-the-Middle (MITM) attacks in Android [46]. Fahl *et al.* found that 1,074 apps in their analysis of 13,500 most popular free apps may be potentially vulnerable to MITM attacks. Rosen *et al.* propose AppProfiler which uses a mapping of API calls and fine-grained privacy related behaviors and static analysis to come up with high-level profiles of application behavior. The authors have shown that the use of profiles has substantial impact on user's understanding of applications [115]. While static analysis is useful in detecting potential software problems, it also has limitations. Moser *et al.* offer techniques to bypass static analysis detection [100].

Dynamic Analysis: Dynamic analysis requires running app binaries along with a combination of instrumenting and/or monitoring techniques. In 2010, the Wall Street Journal studied 101 popular smartphone applications for Android and iOS by monitoring network transmissions while interacting with applications [126]. The authors showed that 56 of the applications sent the phone's unique ID to third party servers without user consent and that 47 of the applications transmitted other sensitive information such as device location and information about the user, e.g., age, gender.

Researchers have explored approaches to automate app monitoring and instrumentation. Enck *et al.* proposed TaintDroid to identify application sensitive transmissions. TaintDroid performs dynamic analysis through tainting data coming from sensitive sources, e.g., GPS module, the user's contact information. The authors studied 30 popular apps from Google Play with TaintDroid and found 15 sharing location with advertisers and another 7 apps sharing the phone's identifiers beyond the user's device. Dynamic analysis has its limitations. For instance, TaintDroid can only monitor explicit data flows and attempts by a malicious developer to use implicit data flows to circumvent TaintDroid detection. MockDroid [20] and TISSA [137] use TaintDroid to evaluate their effectiveness. AppFence adds enforcement policies to TaintDroid [79]. Yang *et al.* integrated dynamic analysis to understand why applications require certain permissions [134]. Specifically, the authors asked crowd workers to compare screenshots of apps with and without access to certain permissions and summarized the differences to understand why applications requested permissions, for instance, using resources for serving ads or providing functionality.

A number of works propose a combination of static and dynamic analysis along with dynamic interaction with the UI to analyze apps. AppScanner, by Amini *et al.*, proposed this kind of analysis [9] and offered a preliminary version of an algorithm to traverse apps while monitoring sensitive app behav-

ior. Gort is an implementation of many of the concepts proposed in AppScanner. SmartDroid uses a similar combination of triggering UI events and performing dynamic analysis [136]. Rastogi *et al.* developed AppsPlayground which also performs a combination of static and dynamic analysis by triggering UI events. AppsPlayground performs sensitive API monitoring in addition to Kernel-level monitoring. Wei *et al* have presented ProfileDroid, a multi-layer profiling of Android Applications which consist of static analysis along with user, OS, and network profiling [131]. In this work, users manually interacted with apps while their interactions were monitored by the profiling tools. Lee *et al.* have developed a traversal algorithm, however, the authors use the traversal approach to evaluate user interface properties for vehicular applications rather than privacy purposes [92].

Gort uses TaintDroid to monitor application sensitive transmissions. As such, the limitations imposed by TaintDroid to detect suspicious behavior are shared by Gort. For an overview of limitations as well as techniques to bypass information flow tracking refer to work by Cavallaro *et al.* [29]. Sarwal *et al.* outline techniques to bypass TaintDroid [118]. While Gort faces these same challenges, Gort's main focus is to inform privacy analysts about app behavior of mobile apps in general, especially in cases where the apps are not intentionally malicious. Gort also presents sensitive behavior in the context of the app by capturing the associated app state and screenshots, whereas TaintDroid flags all potential problems without additional insight about where in the app the privacy leaks occur. Furthermore, Gort uses crowdsourcing to find whether privacy sensitive app behavior is legitimate and desirable to the user, whereas TaintDroid only detect the existence of sensitive app behavior.

Cloud-based Analysis: Shabati *et al.* propose Andromaly to send device features such as resource usage and user activities to a central server for intrusion detection analysis [119]. Paranoid Android creates a clone of an Android phone in the cloud and uses a proxy that prevents data upload to servers when the user is not using the device [111]. Crowdroid crowdsources intrusion detection based on sys calls used by applications [27].

2.2.3 Information Presentation

The focus of this thesis is on mobile app analysis and also better tools for privacy analysts. Another body of work focuses on information presentation and user interfaces to better convey privacy implications of using apps to end-users. For the purpose of completeness, some of the work in this domain is included here. Tam *et al.* studied the usability of permission screens looking at Facebook permissions in a Microsoft Research tech report [123]. Their work focused on how different presentations of resources could help people make better decisions regarding the use of applications. They found the use of icons to depict sensitive resources as more effective than using textual descriptions.

A number of works focus on the Android permission system. Specifically, Kelley *et al.* conducted semi-structured interviews with Android users and found that users paid limited attention to permission screens and in general had a poor understanding of what the permissions meant [87]. Similarly, Felt *et al.* conducted Internet surveys and lab studies to find that the Android permission system did not help users make correct security decisions [54]. Chin *et al.* performed an interview study where they found smartphone users are more concerned about their privacy on smartphones rather than laptops while banking online [32]. Jung *et al.* performed a field study and found that the users were surprised by the amount and frequency of data leaving their phones [85]. Similarly, Balebako *et al.* found that users are surprised by the frequency and destination of data leaks [16]. The authors also evaluated two types of interfaces that present data leakage to users. Egelman *et al.* found that 80% of users are willing to receive targeted advertisement regardless of permissions used by the application as long as it saved them $0.99 [41]. Felt *et al.* provide guidelines for using several permission-granting mechanisms [52]. Finally, Kelley *et al.* proposed to improve Android's permission screen by putting the privacy facts inline with the app description [88]. Lin *et al.* used crowdsourcing to capture users' mental models and expectations regarding mobile app behavior to offer better privacy summary interfaces for mobile apps [93]. These works mostly focus on the smartphone end-user, whereas Gort focuses on addressing the needs of privacy analysts.

2.2.4 Crowdsourcing

Crowdsourcing has been growing quickly both as a topic of research and as a tool for research. Amazon's Mechanical Turk (MTurk) is currently the most popular crowdsourcing platform and is the one used for this work. Most tasks on Mechanical Turk do not require special skills and tend to involve small payments. Crowdsourcing has been successfully used in a number of projects, for example, selecting labels for images [130] and conducting user studies [89].

Kittur and Kraut have explored crowdsourcing with respect to creation of Wikipedia articles [90]. Crowdsourcing has also been used in the past to evaluate visualization design [75]. Soylent is perhaps one of the most complex crowdsourcing systems. Soylent uses multiple ways of structuring tasks and organizing people so as to yield reasonably good results for word processing [21]. The authors have shown that certain kinds of design patterns in organizing workers could yield good results. For instance, Soylent introduced the Find-Fix-Verify pattern to bring the wisdom of crowds to word processing. Crowdsourcing has also been used in the mobile computing domain. Yan *et al.* use crowds to perform real-time image search on mobile phones [133]. Amini and Yang propose CrowdLearner to automatically generate mobile recognizers for mobile sensory input for developers and researchers [8]. For an overview of human computation refer to Quinn and Bederson's taxonomy of human computation [112].

Researchers have studied automating the creation of complex crowd sourcing tasks. Little *et al.* present a toolkit, TurKit, to explore human computation algorithms on Mechanical Turk. Kittur *et al.* offer CrowdForge [91] to break down complex tasks using a process similar to Map Reduce [36]. Little *et al.* have also experimented with using iterative and parallel human computation processes [94]. They have found that iteration generally increases the average quality of responses. However, for tasks that require a high variability of responses, a parallel process can be more effective.

Prior work has explored how participants game crowd-sourcing platforms and how to mitigate this problem [39, 89]. Downs *et al.* identify groups who tend to perform poorly [39]. Kittur *et al.* recommend the use of explicitly verifiable questions to identify poor task responses, designing tasks such that completing tasks accurately takes as much or less effort than non-obvious random or malicious completion, and finally having multiple ways to detect suspect responses [89]. Ipeirotis *et al.* offer techniques that estimate the quality of crowd workers allowing for rejection and blocking of low-performing workers and spammers [83]. Rzeszotarski and Kittur propose using implicit behavioral measures to predict task performance [117].

Perhaps, the first use of crowdsourcing in the mobile privacy and security domain was proposed by the author to gauge end-user expectations regarding the privacy implications associated with using mobile apps. Refer to Appendix A for the original proposal. The details of this proposal are further described in a tech report by Amini *et al.* [9]. Work by Lin *et al.* was a direct result of this proposal, which gauged crowd workers' expectations and comfort with mobile apps using the Mechanical Turk platform [93]. This work had three important conclusions. First, that users' comfort with an app is directly correlated with their expectation of how the app behaves. Second, that providing more information about how an app uses a sensitive resource or permission to users increases the comfort with the application behavior. Third, that crowd workers on Mechanical Turk have a hard time trying to figure out why an app uses certain resources. However, considering the size of the app markets, Lin's approach would not scale as data presented to the crowd for analysis was extracted manually.

The work presented by this thesis automates the end-to-end process to obtain crowd opinion regarding mobile app privacy. Moreover, the breakthrough crowdsourcing approach and the *extraction-verification-justification* paradigm offered by the author enable Gort to find whether an app's use of sensitive information is legitimate and desirable by users through crowdsourcing. While crowd workers cannot explicitly answer whether the use of a certain resource is legitimate, using the paradigm, the crowd can implicitly answer this question. The implications of this work are also important to the field of crowdsourcing, as Gort presents how a non-technically oriented crowd can implicitly answer questions that require technical expertise.

Another work that has used a crowdsourcing approach to answer technical questions is Communitysourcing by Heimerl *et al.* [76]. The authors designed a system to grade questions from a computer science class by offering snack incentives to students using a vending machine. The technique is referred to as Communitysourcing as it targets tasks at a community of users with a desired expertise, e.g., the expertise to verify students' answers to computer science related questions. Gort's crowdsourcing approach is different as it uses a non-technically oriented crowd to implicitly answer technical questions.

Crowdroid is another work that relies on crowdsourcing in the mobile privacy domain [27]. Specifically, Crowdroid obtains traces of app behavior using crowdsourcing, which are then used to differentiate between benign apps and those containing malware. Another work, ProtectMyPrivacy, has developed a crowdsourced recommendation-engine which collects protection decisions by the users and makes privacy recommendations for apps [1]. While both these works rely on crowdsourcing, they focus on different aspects of the mobile app ecosystem and have a different set of inputs and outputs.

This thesis builds on past crowdsourcing research in several ways. First, it develops new ways of applying crowdsourcing to computer privacy. Second, it looks at new ways of combining automated techniques with crowdsourcing, including, capturing perceptions of privacy for a given app (coarse granularity) as well as specific screens and sequences of screens (fine granularity). Third, this thesis investigates new ways of doing effective division of labor between the automated system and the crowd, as well as ways of aggregating crowd responses to yield semantically useful results.

3 Gort App Analysis Overview

This chapter provides an overview of Gort and its various components. The overview is presented for the latest version of Gort, namely, v2. Gort is comprised of multiple subsystems which work together to evaluate mobile app privacy. Gort aims to achieve two goals:

1. Provide an end-to-end system to perform mobile app privacy analysis while taking into account context and semantics. This end-to-end system is called the *Analysis Engine*.
2. Offer an easy to use GUI tool for privacy analysts to analyze a set of applications (order of tens) or a single application at a time. This tool is referred to as the *Gort GUI*, *Gort tool*, or the *GUI tool* throughout this thesis.

The Gort GUI is a tool for privacy analysts to view the results generated by the Analysis Engine. An overview and also the details of the tool are covered in §7. This section provides an overview of the Analysis Engine to show how different components of the system fit together and the design rationale. For the purposes of this work, the focus will be on the Android operating system. This decision was made based on the open source nature of Android as well as the availability of tools and extensions for Android (in particular, TaintDroid [43], and TEMA [122]). For an overview of Android, refer to §2.1.2.

3.1 Analysis Engine

The Analysis Engine is comprised of three subsystems which perform static, dynamic, and crowd analysis. The engine analyzes applications in three steps, where each one of the aforementioned subsystems processes the application. These processes are always executed in the same order, namely: static, dynamic, crowd analysis. Figure 3 provides an overview of the Analysis Engine.

3.1.1 Static Analysis

In the first stage of the analysis, the engine performs static analysis. Static analysis refers to analysis that does not require the running of the application. Gort runs static analysis first, as it is more time and resource efficient. Furthermore, performing static analysis first provides some information about the app early on in the process. The bulk of static analysis is conducted using Androguard [37], which provides a set of APIs for performing analysis on Android apps. In addition to this analysis, at this stage, Gort also downloads application metadata from Google Play, and determines the output of the static heuristics.

Extract App Components: While performing static analysis, Gort extracts application components using Androguard. The results are used to guide the exploration of the application in the dynamic analysis

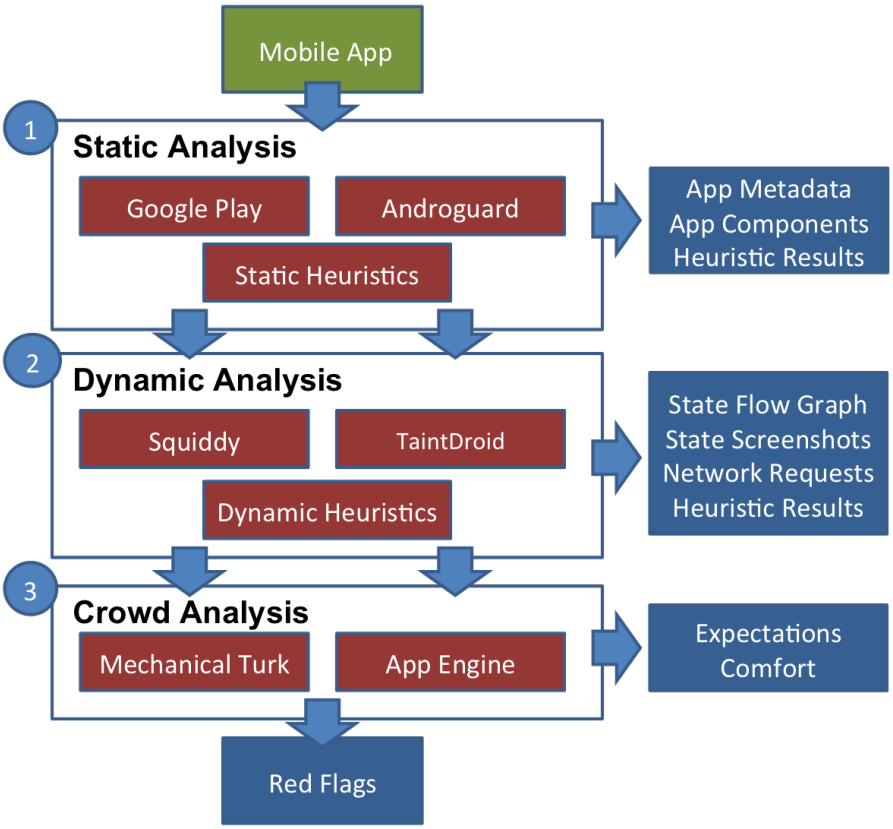


Figure 3: The Analysis Engine scans applications in three stages. First, it performs static analysis which reads application components and does basic static analysis through the use of Androguard, downloads app metadata and screenshots from Google Play, and performs static heuristics that can be determined without running the application. In the second stage, Gort performs dynamic analysis using its traversal algorithm called, *Squiddy*, takes screenshots of the states, and monitors sensitive network transmissions through TaintDroid. Also in the second stage, Gort generates a *State Flow Graph* (SFG) of the application and determines the output of the dynamic heuristics. In the third stage, Gort uses the app metadata from stage one and results from stage two to produce human intelligence tasks for crowdsourcing. The output of the final stage are the crowd's expectations and comfort levels regarding application behavior and red flags about the application based on the crowd's expectations and comfort level.

stage and also used to only perform crowd analysis on application components which are part of the app package. Specifically, the following app components are extracted using Androguard:

- Package Name: Helps identify apps in the Google Play store (e.g., com.yelp.android)
- Permissions: Set of permissions requested by the application
- Application Components: Activities, Services, Providers, Receivers
- Libraries: Third-party libraries used by the application
- SDK: Minimum and target SDK for the application

In addition to extracting components, Gort also performs elementary static analysis to find the use of the following coding techniques with which the application may execute code not bundled as Java code in its application package:

- Dynamic Code: Androguard checks for the use of DexClassLoader[6]. DexClassLoader is a class loader that loads classes from .jar and .apk files containing a classes.dex entry. This can be used to execute code not installed as part of an app.
- Native Code: Android apps can load code written in C/C++ by using the Native Development Kit (NDK)[7]. This enables developers to reuse code that they have already written, e.g., for other platforms. Native code can also be used to boost application performance for resource intensive applications. Nevertheless, the use of native code increases the complexity of the app. Furthermore, some existing analysis approaches do not check native code for sensitive behavior [43].
- Java Reflection: Java Reflection is used to examine or modify the runtime behavior of applications running in the Java Virtual Machine[8]. Using this feature, Android applications can load Java classes at runtime, which poses a security risk.

Download App Metadata from Google Play: During the static analysis stage, the Analysis Engine also downloads app metadata from Google Play. Specifically, the engine extracts the app name, screenshots, developer, category, and description from the Google Play website. The aforementioned information is necessary for two reasons. First, the app name and description are used during the Crowd Analysis stage, where they are presented to crowd workers. Second, the app metadata extracted is conveniently available to analysts in the Gort tool.

Run Static Heuristics: The static heuristics' outputs are determined during the static analysis phase. The static heuristics typically take less time to compute than the dynamic heuristics. For additional information regarding the heuristics refer to §5. The key idea for executing a portion of the heuristics at this stage is that it provides an overview of app behavior and potential privacy problems without completing all of the analysis stages. This approach is also helpful with respect to the Gort tool in that privacy analysts can get an idea of the app's potential privacy problems without going through the entire analysis process.

3.1.2 Dynamic Analysis

Gort performs dynamic analysis to learn about apps' sensitive transmissions and also to determine the output of the dynamic heuristics. For more information about heuristics refer to §5. Dynamic analysis is

[6] `http://developer.android.com/reference/dalvik/system/DexClassLoader.html`
[7] `https://developer.android.com/tools/sdk/ndk/`
[8] `http://docs.oracle.com/javase/tutorial/reflect/`

the second stage of analysis, since it produces the input to the next stage, namely, crowd analysis. It is also less costly than crowd analysis in terms of both time and resources. The dynamic analysis runs apps in an isolated VM in order to minimize interactions with other third-party applications which would otherwise curtail the quality and accuracy of the dynamic analysis results.

Traversing Apps to Trigger App Behavior: The core of dynamic analysis depends on interacting with the app's user interface elements to trigger behavior and explore various states of the app while monitoring sensitive behavior. Gort uses an app traversal algorithm, *Squiddy*, to interact with the app and TaintDroid [43] to monitor the app's sensitive transmissions. For an in-depth discussion of the traversal algorithm refer to §4.

During the dynamic analysis, Squiddy interacts with app UI elements to explore and capture various states of the app. As Squiddy traverses the app, it also takes screenshots of the app, stores the UI components that it discovers, maintains graphs describing the traversal exploration and discoveries, and also records the sensitive transmissions reported by TaintDroid in the form of *TaintLogs*.

Post Processing Traversal Results: Once Gort has finished running the traversal algorithm on an app it performs post processing. The post processing step performs several key tasks, specifically:

1. For each sensitive transmission:
   (a) Extracts the sensitive transmission IP address
   (b) Finds the associated URL and registrant information for the IP
   (c) Determines the types of sensitive information transmitted by extracting the taint tag (e.g., location, contacts, unique device identifier)
2. Determines the associated app state for the taints recorded
3. Produces a visualization of the recorded app traversal graph for use by the Gort Tool (§7). This step visualizes the State Flow Graph that is presented to privacy analysts using the Gort tool.
4. Determines the output of the dynamic heuristics

The output of the post processing is important for two reasons. First, the post processing maps sensitive transmissions to specific states in the app. This mapping is essential for the crowd analysis step to present the correct state with its associated resource usage to the crowd, and also for the Gort tool to present the right resources and sensitive transmissions to the privacy analysts. Second, the post processing determines the output of the dynamic heuristics which can be used to get a high-level overview of the app, filter out apps based on sensitivity, and convey more information regarding the app's potential privacy problems to privacy analysts using the Gort Tool.

3.1.3 Crowd Analysis

Gort performs crowd analysis to determine whether the sensitive resources used for the application are justified by the application context and semantics. In this phase of the analysis, Gort uses the app metadata downloaded from Google Play and the sensitive transmissions with the associated screenshots to create Mechanical Turk Human Intelligence Tasks (HITs) to evaluate the app's behavior. The intermediary outputs of this stage are the crowd expectations and comfort levels, which are then used to produce red flags about privacy-intrusive app behavior. For details of the crowd analysis refer to §6.

4 Automating App Traversal to Explore and Trigger App Behavior

This chapter presents how Gort traverses apps to trigger and analyze their behavior at runtime. Traversing an app refers to the process of visiting and inspecting each state of an app. The chapter opens with a discussion of why traversing apps is necessary. It then delves into the details and evaluation of two versions of Gort's traversal algorithm, Squiddy[9]. Squiddy v1 helped explore the space and highlight some of the challenges for app traversal. Squiddy v2 mitigates some of the major challenges and provides improved app coverage. This chapter also discusses the traversal challenges, and offers insights to guide the design of future traversal algorithms.

4.1 Why Traversal Is Necessary for App Analysis

In order to provide a thorough analysis of mobile apps, it is important to evaluate them using both static and dynamic analysis techniques. Static analysis techniques consist of analysis methodologies that do not require one to run an app in order to analyze it. While this feature of static analysis techniques is appealing, effectively employing such techniques is non-trivial [44]. Furthermore, static analysis techniques may not be able to capture the app's runtime behavior. Finally, while static analysis can scale the detection of potential privacy leaks, the challenge lies in determining which user interactions and inputs trigger privacy leaks and whether the privacy leaks are necessary and beneficial to the user, i.e., the user is knowingly leaking sensitive information for a desired feature or service.

Privacy analysts' traditional approach has been to interact with a mobile app while monitoring its sensitive transmissions to trigger app behavior and determine the legitimacy of privacy leaks [78, 81, 126]. The typical inspection process employed by analysts is manual and rather tedious. It consists of routing app network transmissions through an isolated network environment or a proxy, downloading and testing a single app at a time, interacting with the app manually to inspect its behavior, and making sense of a variety of abstruse resulting outputs [78, 81, 124, 125]. Such approaches consume valuable analyst time and generally require expertise in setting up a proper analysis bench. As a result, the level of expertise and effort required do not scale well and pose as a barrier to effectively evaluating mobile app privacy.

One way to help privacy analysts evaluate mobile apps is to offer an automated solution to interact with apps and monitor their behavior. At the time that I set out to provide such a solution, there did not exist any automated mobile application traversal techniques. Analysts interacted with applications either manually or through using scripting languages. While a number of tools existed to support creating scripts

[9]Squiddy is named after robot Sentinels from film, *The Matrix*.

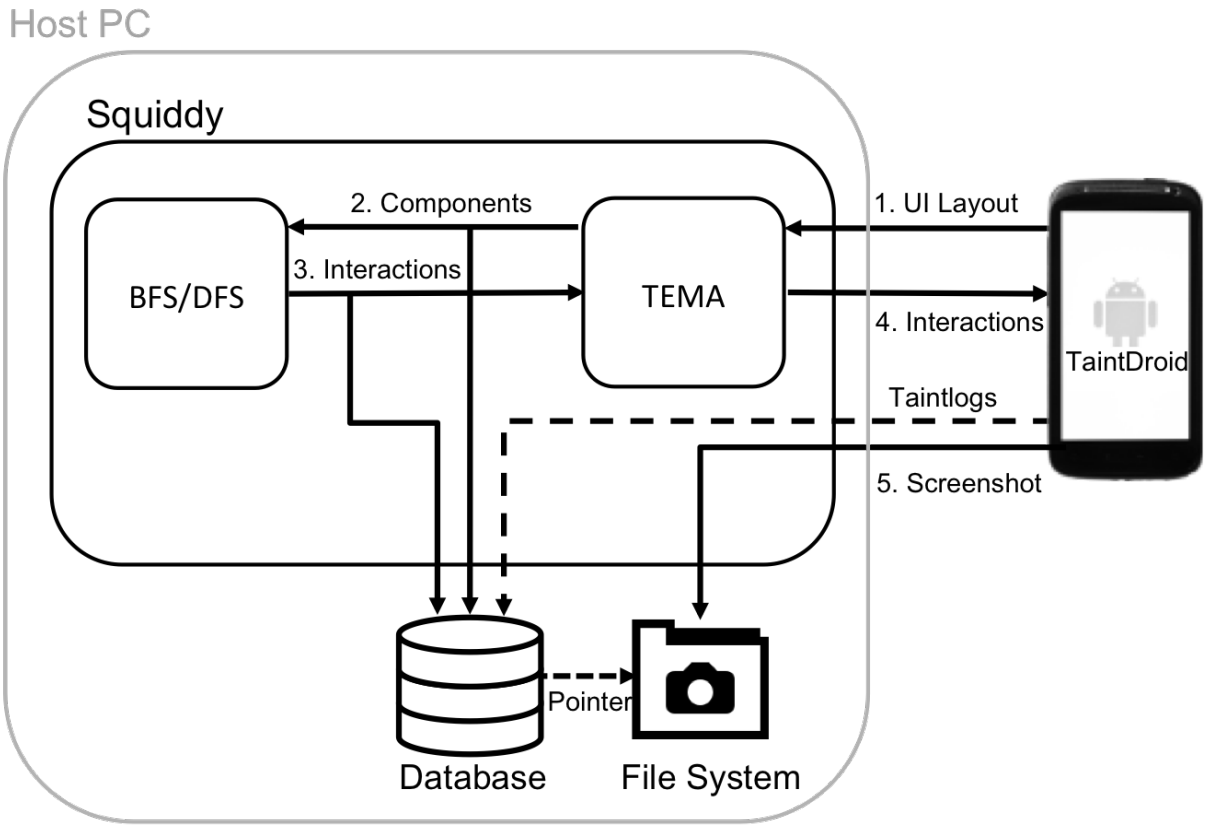


Figure 4: Gort automatically interacts with apps using its traversal algorithm, Squiddy, while monitoring sensitive privacy leaks through TaintDroid. Squiddy runs a search algorithm that interfaces with TEMA to discover and interact with app UI elements. TEMA is a testing framework for Android, which supports reading an app's UI elements via a Document Object Model (DOM) and sending interactions to them. While traversing apps, Squiddy also records the logs from TaintDroid, referred to as TaintLogs, and takes screenshots. Steps 1-5 execute continuously in a loop until the app is completed traversed or the traversal time limit (60 minutes) is reached.

to perform UI interactions (e.g., Robotium[10], Scirocco[11], Android Money Runner[12], and TEMA [122]), the existing solutions required expert knowledge and imposed a learning period. Since then, a number of traversal approaches have been proposed by the author [9], and by Lee *et al.* [92], Rastogi *et al.* [114], and Zheng *et al.* [137]. Lee *et al.* use their traversal algorithm to evaluate user interface properties for vehicular application rather than privacy purposes. However, all algorithms use a combination of reading an app's user interface and interacting with its UI elements.

[10] `http://code.google.com/p/robotium/`
[11] `http://code.google.com/p/scirocco/`
[12] `http://developer.android.com/tools/help/monkeyrunner_concepts.html`

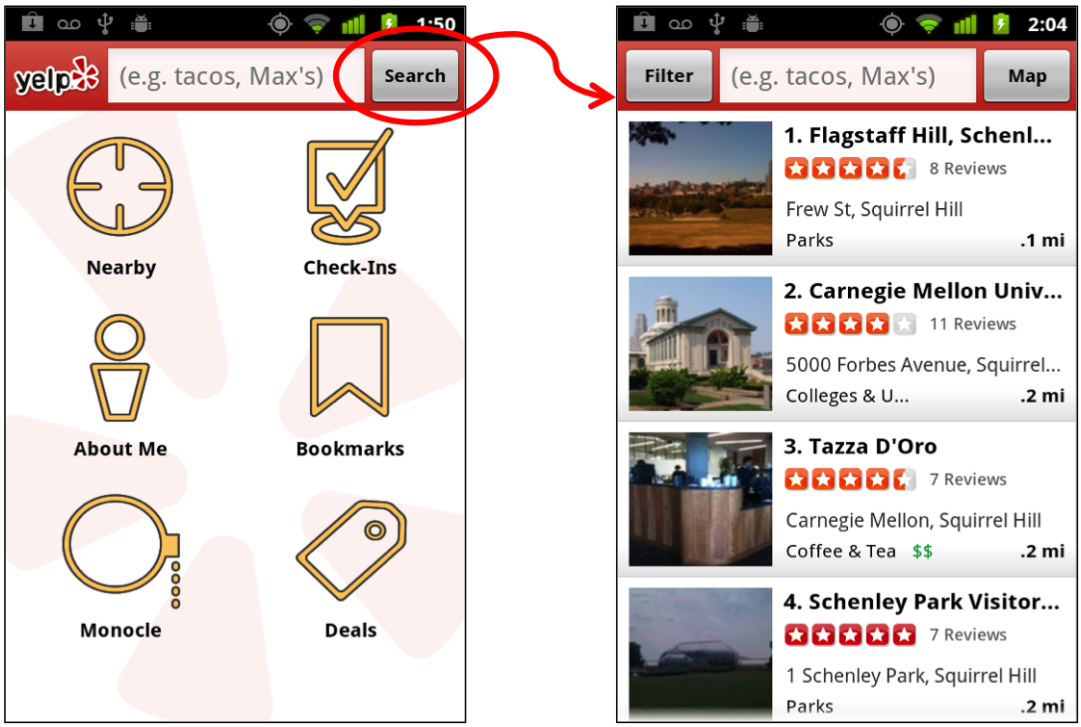


Figure 5: Squiddy interacts with GUI elements to explore and discover different states of an app. This figure shows an interaction performed by an actual run of the Squiddy v1 algorithm to explore the Yelp app. The first version of Squiddy interacts with GUI elements to find Android app activity components. While the second version of Squiddy uses the same approach, it attempts to discover and explore app states, which may be part of the same activity or different ones. Traversing for app states instead of activities provides better traversal coverage.

4.2 First Iteration of Squiddy

The first version of Squiddy automatically interacts with a given mobile app to discover its different activities, takes screenshots of the activities, and monitors the sensitive transmissions corresponding to specific activities. Figure 5 presents a sample interaction. I designed Squiddy to work out of the box, without the need for an expert to script specific interactions. In addition to traversing the app and taking screenshots, Squiddy v1 monitors what resources are being used by the app in each activity. One could monitor app resource usage through a network analyzer as was done by the Wall Street Journal study of mobile apps [126] and Hunt's independent expert study of app data usage [81]. However, the issue with just using network analyzers is that unless the data is unencrypted, it is not possible to determine what sensitive information is being sent out. Also, parsing the mobile apps network requests are non-trivial [14]. As such, Squiddy v1 relies on dynamic information tracking, specifically TaintDroid [43], to monitor app resource usage. TaintDroid logs the occurrence and type of sensitive information transmitted. Figure 6 shows sample TaintDroid logs from the Yelp and Horoscope apps.

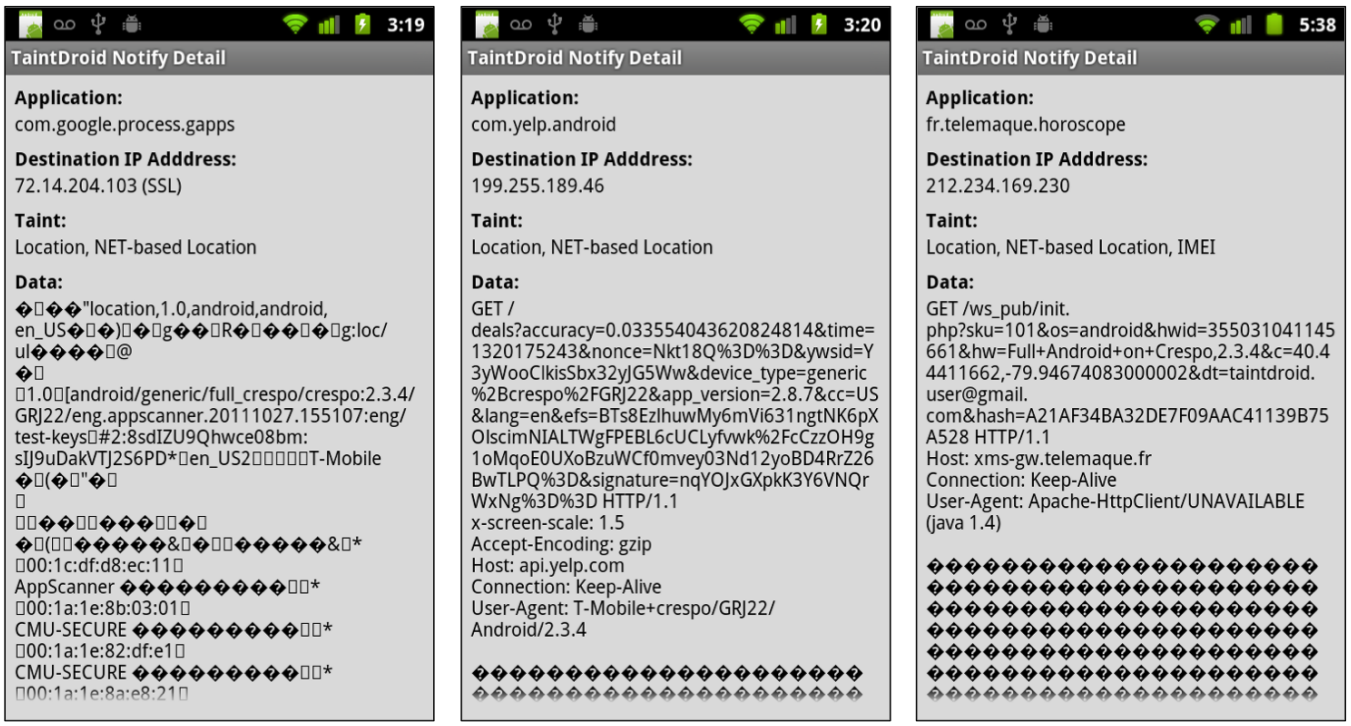


Figure 6: While Squiddy automatically traverses apps by interacting with their UI, it also monitors their privacy leaks using TaintDroid. TaintDroid uses dynamic information flow tracking techniques to capture app privacy leaks. This figure presents samples from the Yelp (left, middle) and Horoscope (right) apps where a user's location is shared with external entities. The Horoscope app also sends out the device IMEI which is a unique device identifier. Squiddy records the privacy leaks reported by TaintDroid through accessing the Android device logs.

4.2.1 Squiddy v1 Algorithm Details

Squiddy v1 traverses apps by running a *breadth-first search* (BFS) traversal on top of TEMA [122] and monitoring the use of sensitive information by instrumenting apps using TaintDroid [43]. Using TEMA, Squiddy obtains UI widgets available on the current screen (using Android's *ViewServer* service) and simulates interaction with the widgets (using Android's *monkeyrunner* service). The ViewServer outputs the layout of the current screen on the device in the form of a Document Object Model (DOM). TEMA reads and parses this object model to obtain the widget properties for each element, e.g., class type, coordinates, width, height. The properties are then used to find and interact with components on the screen. Figure 7 presents an example of the ViewServer output viewed using the Android *hierarchyviewer* tool.

Squiddy interacts with Android apps by clicking on interactive elements of the apps (Figure 5). Squiddy v1 is activity-based, meaning it attempts to discover new app activities rather than explore various states

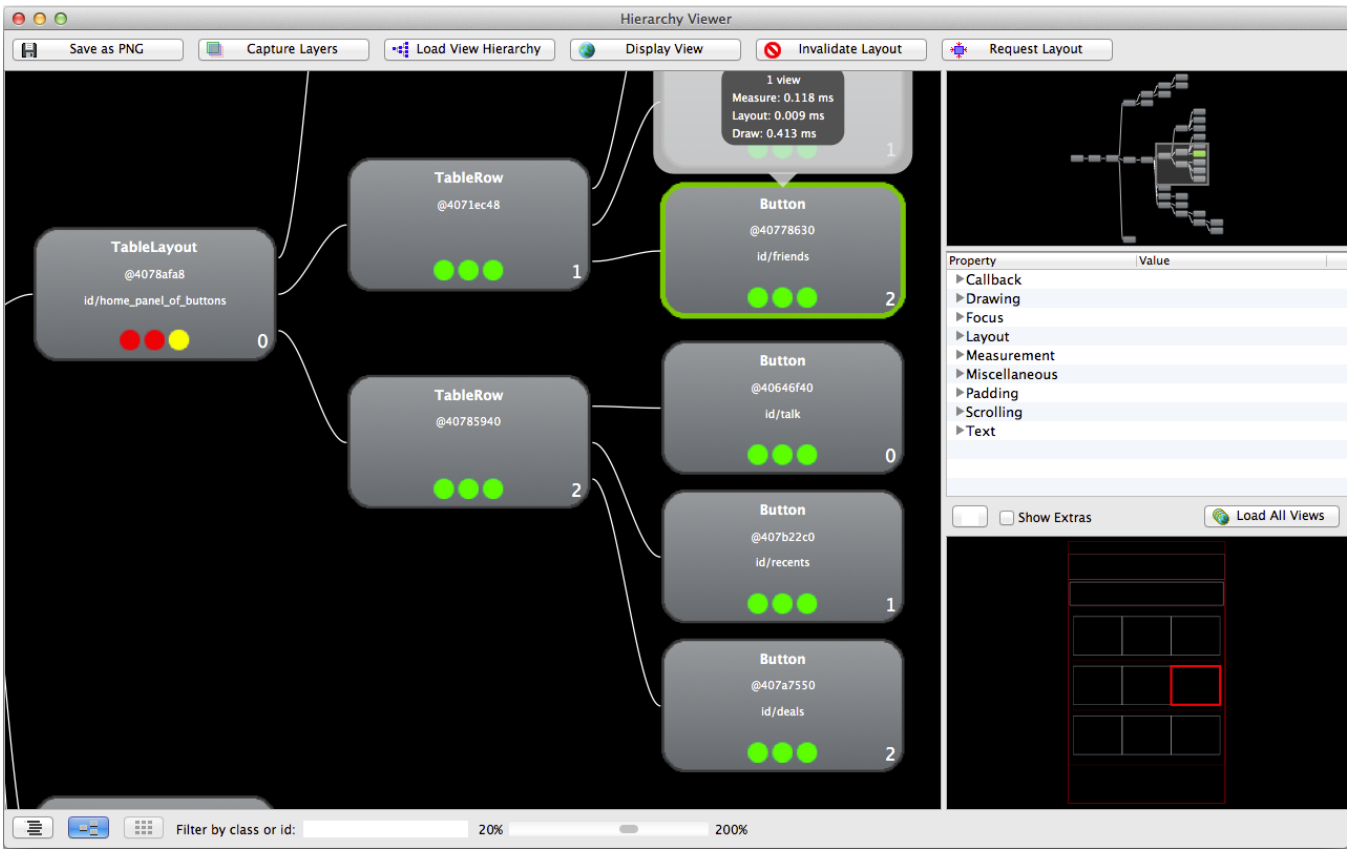


Figure 7: The Android hierarchyviewer tool presents the GUI layout hierarchy on the screen of the connected device or emulator. The layout information is sent from the device to a host PC as a Document Object Model (DOM). The figure here presents the hierarchy for the main screen of the Yelp app.

of an activity in depth. An activity represents a single screen with a user interface[13]. As Squiddy discovers more activities of the app, it continues digging deeper into the app. During a post-processing stage, an *Activity Flow Graph* (AFG) diagram of the app, which could be viewed and interacted with using the GORT GUI, is generated programmatically using Graphviz [71]. Figure 8 shows a sample AFG for the Yelp app. The TaintDroid logs of sensitive transmissions are recorded with timestamps during the traversal. Squiddy records the transmissions by accessing the Android log while interacting with the application. The post traversal processing also maps the captured privacy leaks with their corresponding activities using the timestamps.

4.2.2 Squiddy v1 App Coverage

The first version of Squiddy was evaluated through estimating the algorithm's app coverage. All traversals are set to timeout after 60 minutes as the traversal would start to experience diminishing returns and

[13] `http://developer.android.com/guide/components/fundamentals.html`

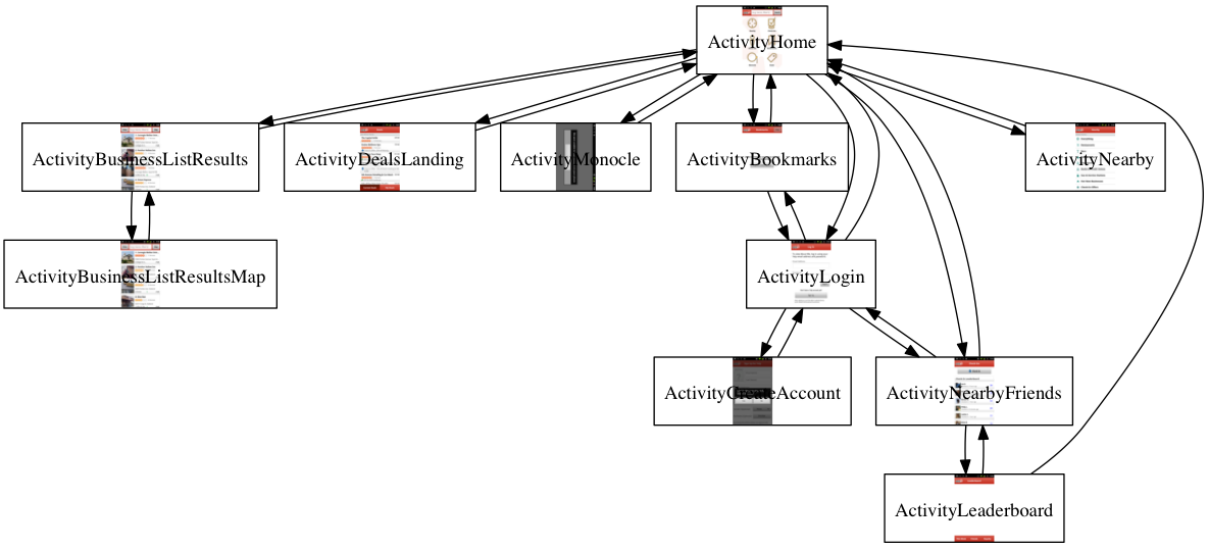


Figure 8: After Squiddy v1 traverses an app, it generates an *Activity Flow Graph* (AFG) showing the app's activities and their relationship to one another. The AFG above shows major screens in the popular Yelp application. Squiddy v1 also captures actual screenshots of screens that users would see. A thumbnail version of the corresponding app screen is shown inside of each node.

the approach would not scale. Squiddy v1 focuses on exploring and discovering Android app activity components. However, while using Squiddy v1, it was quickly discovered that some Android apps use multiple UI layouts in the same activity component. The use of tab views also complicates app traversals as UI elements would change on different tabs. I refer to this design pattern as *activity stuffing*. However, as Squiddy v1 is activity-based, app coverage is computed in terms of activities. This coverage is a preliminary estimate for three reasons. First, it is only UI based and not based on source code coverage or branch coverage. Second, it may overestimate coverage as it only looks at activities and does not account for activity stuffing. Third, it may underestimate coverage as the number of the app's total activities was found through the app manifest, which may declare activities that are not actually executed by the app.

Based on the aforementioned definition of coverage, Squiddy v1's coverage was evaluated using 19 apps. The mean and standard deviation for coverage were 13.9% and 13.0 percentage points, respectively. The minimum coverage was 2.3% for the EBay app and the maximum was 50.0% for the Wikipedia app. Table 1 presents the results for the percent coverage using Squiddy v1.

4.3 Traversal Hurdles and Lessons from Squiddy v1

Traversing apps to explore app states and trigger behavior is non-trivial. While the average coverage for Squiddy v1 was 13.9% of an app's activities, Squiddy v1 was extremely informative in exposing the challenges of traversing apps and identifying areas for improvement. The insights from Squiddy v1 helped

App	Discover	Declared	Coverage (%)	Time (s)
Best Parking	1	14	7.1	224
Bible App	4	76	5.3	1750
CityShop	1	17	5.9	176
ColorNote Notepad	1	20	5.0	392
EBay	3	132	2.3	511
GasBuddy	4	42	9.5	433
G. Finance	1	8	12.5	95
G. News & Weather	1	6	16.7	1710
G. Search	1	8	12.5	70
Groupon	3	70	4.3	3600
KBB.com	8	18	44.4	3600
QR Code Reader	1	4	25.0	374
Relax Timer	1	6	16.7	1379
Stopwatch & Timer	1	9	11.1	258
Tag	1	7	14.3	282
Tumblr	2	30	6.7	3600
Weather Channel	1	16	6.3	130
Wikipedia	1	2	50.0	115
Yelp	12	134	9.0	1768

Table 1: This table presents percent coverage of apps using the traversal algorithm from the first version of Squiddy. Coverage is calculated as the number of activities discovered by Squiddy divided by the number of activities declared in the application manifest. The mean and standard deviation for coverage were 13.9% and 13.0 percent points, respectively.

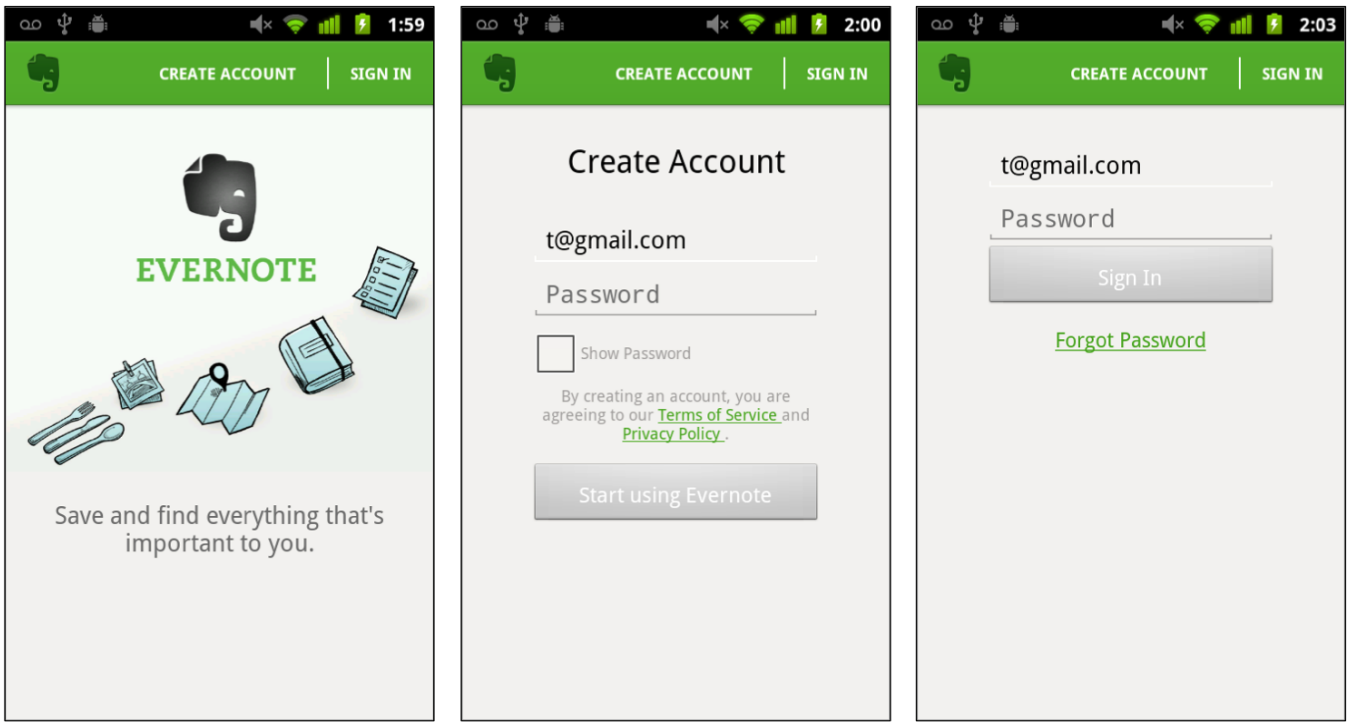


Figure 9: It is not uncommon for single Android activity components to present several UI layouts. This figure illustrates the main activity of the Evernote app which has several UI layouts accessible through swipe interactions or the top navigator bar. Squiddy v1 reads the layout of each activity only once when it first visits the activity. As such, activities that have multiple layouts would only be traversed with respect to the layout which is visited first. This also results in problems when the traversal arrives to an activity for the second time to be introduced to a completely new layout.

introduce solid coverage gains for Squiddy v2. In this section, I outline the lessons learned from Squiddy v1. For an overview of the challenges and potential solutions, refer to Table 2.

4.3.1 Activity Stuffing

As previously mentioned, some Android apps implement more than one UI layout in a single Android activity or use tabs to present multiple views. Since the first version of Squiddy is activity-based, it tries to explore and discover activities based on a single read of the activity screen upon first visiting the activity. This approach results in a partial discover of the app views and hence leads to a reduction in coverage. In extreme cases, the traversal algorithm would arrive at an activity a second time to find a completely new layout. The new layout would confuse the traversal algorithm as none of the prior existing UI elements would be available for interaction. This problem was later resolved in Squiddy v2 through exploring app states rather than activities. In Squiddy v2, activities can have multiple states, and states are read after each interaction. Nevertheless, there exists a trade-off in that screens have to be read after each interaction regardless of the activity being explored. Multiple reads result in a jump in the amount of time spent reading the app rather than interacting with it. Figure 9 shows a layout-based activity stuffing example.

Name	Description	Solution	Impl†	Section
Activity Stuffing	App has more than one UI layout in an activity or uses tab views	Traverse apps based on states rather than activities, similar to AMC [92]	✓	4.3.1
Canvas Based UIs	App has canvas-based custom widgets, which are not recognizable as individual elements	Use grids to interact with different cells of a canvas, similar to Caché [10]		4.3.2
Complex Interactions	App requires complex interactions prior to exhibiting certain behavior (e.g., game leader board only shows after playing an entire level)	Manually call activities and receivers, discovered through static analysis, (AppsPlayground [114])		4.3.3
Lists and Scrollables	Number of elements in a list can dynamically change at runtime. Some lists can have an infinite number of items (e.g., FB Newsfeed)	Limit list interactions for homogeneous lists, similar to AMC [92]	✓	4.3.4
Non-determinism	App has sporadic or random behavior (e.g., sporadically asks the user to rate the app on the Google Play store)	Use design patterns to cancel sporadic dialogs	✓	4.3.5
Special (Physical) Input	Some behavior is only triggered by special input (e.g., credit card swipe for Square, requests Fitbit records)	Guess input or ask a human for input		4.3.6
Special Setup	App requires special setup prior to running (e.g., Pulse app requires the user to select news interests)	Guess input, use scripted input, or ask a human for input		4.3.7
System Apps	App can change system behavior (e.g., turn WiFi on or off)	Closely monitor system status or modify kernel to prevent changes to the system		4.3.8
System Instability	Instrumenting the app de-stabilizes the system (e.g., running TaintDroid instead of the stock Android build)	Use the most stable system available, analyze one app at a time, and traverse quickly!	✓	4.3.9
Time Synchronization	Time of the host PC used for traversal and the device running apps does not match	Use a (simple) time synchronization algorithm	✓	4.3.10
User Authentication	App requires login to expose feature (e.g., Facebook, Twitter, Pinterest)	Manually call activities and receivers, discovered through static analysis or ask human for credentials		4.3.11
Virtual Keyboard	Traversal cannot interact with components hidden behind the virtual keyboard	Recognize keyboard using Computer Vision or modify kernel to report virtual keyboard visibility	✓	4.3.12

Table 2: Shows an overview of traversal challenges, how to mitigate them, and the associated section. †The *impl* column refers to whether a solution was implemented in Squiddy v2.

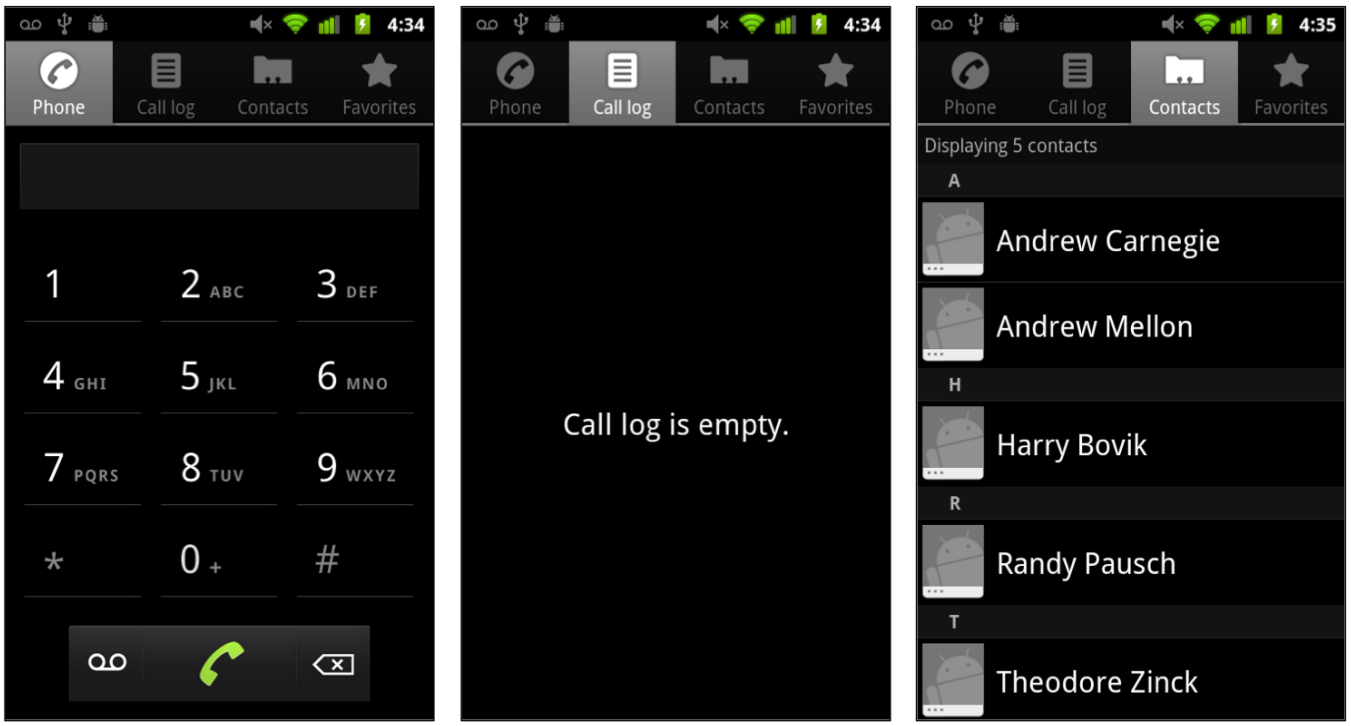


Figure 10: Using Tabs is another design paradigm that leads to a single Android activity having multiple layouts. Such layouts would confuse traversal algorithms that make a single layout assumption about activities, such as the first version of Squiddy.

4.3.2 Canvas Based Apps

Squiddy v1 and v2 are both widget based traversal algorithms, i.e., the algorithms only find and interact with default Android UI widgets. Therefore, interaction is limited when it comes to custom widgets designed by app developers, e.g., in game UI elements rendered on a canvas. Map interfaces are another example. Squiddy v1 and v2 would detect the canvas as a single element and provide a single click to the canvas. If the single click does not lead to a state change, the canvas would no longer be explored.

Game apps are quite popular on mobile platforms. As such, a comprehensive traversal algorithm should work around the canvas widget problem. One approach to solve this problem is to place a grid on canvases and explore by providing interactions to the grid cells. Multiple level grids and overlapping grids can also be used to provide better exploration of the canvas. This technique would be similar to how Caché provides input to location-based search engines to prefetch content for an entire geographical region [10]. Figure 11 provides an example of the canvas problem and how a grid based interaction solution could help.

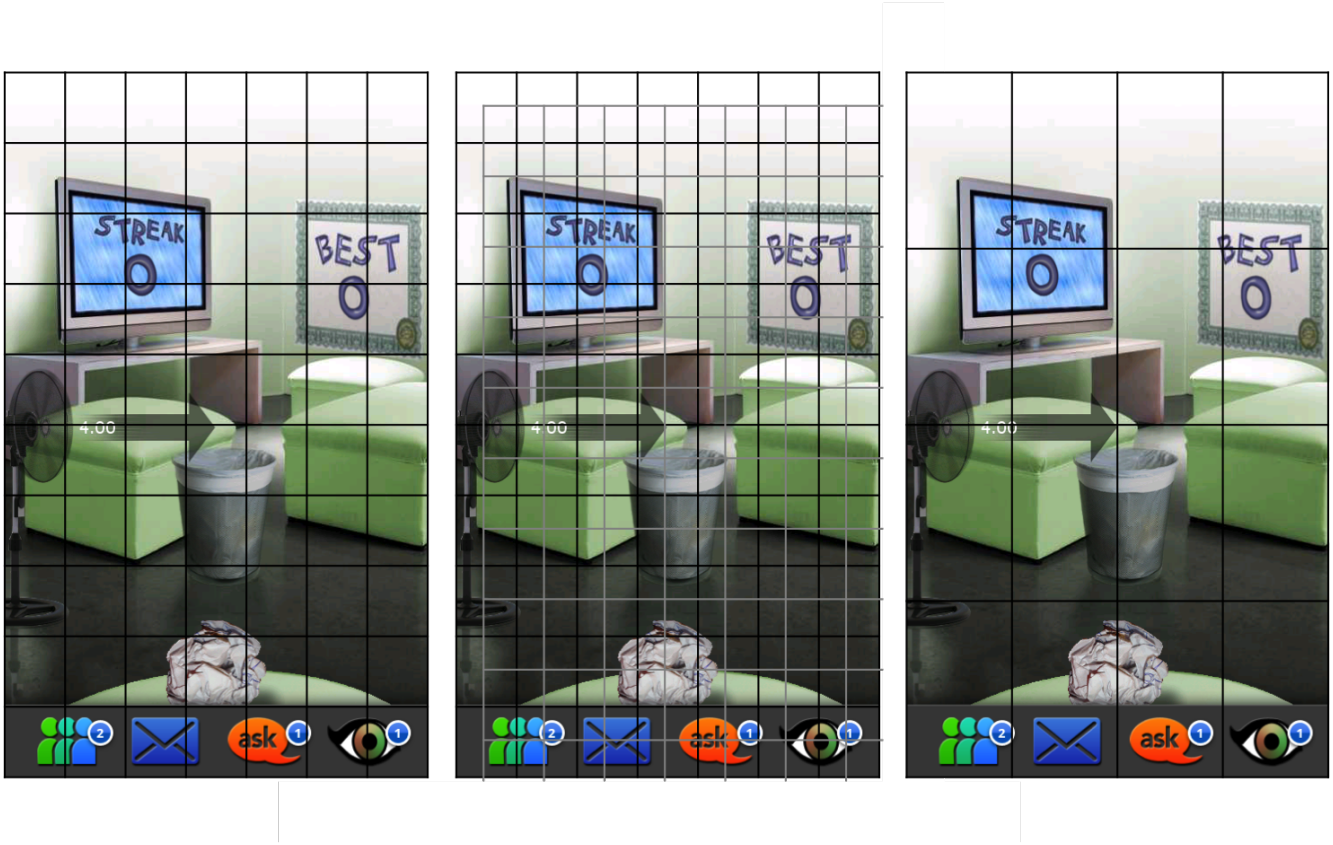


Figure 11: Apps with custom designed UI components, such as games, present a challenge to a UI element based traversal algorithm. A grid based interaction solution may provide a work-around at the cost of additional interactions per element. Grids could be overlapped for better coverage or use a variety of sizes.

4.3.3 Complex Interactions

Some app behavior may only be triggered after a sequence of complex user interactions. Complex user interactions can be something as simple as providing a swipe action to the screen to view a different layout (similar to Figure 9) or as complicated as playing an entire level of Angry Birds[14] to reach a different level or the leader board screen. Unless the traversal algorithm triggers these complex interactions or offers work-arounds to reach unexplored states (e.g., generate events to trigger broadcast receivers [114], or manually open activities), some app states may be beyond the reach of the traversal algorithm.

4.3.4 Lists and Scrollable Areas

Some layout containers, such as lists, allow the developer to dynamically load child items at runtime. While a useful feature for developers, lists and scrollable layout elements make traversals challenging in

[14] `http://www.angrybirds.com`

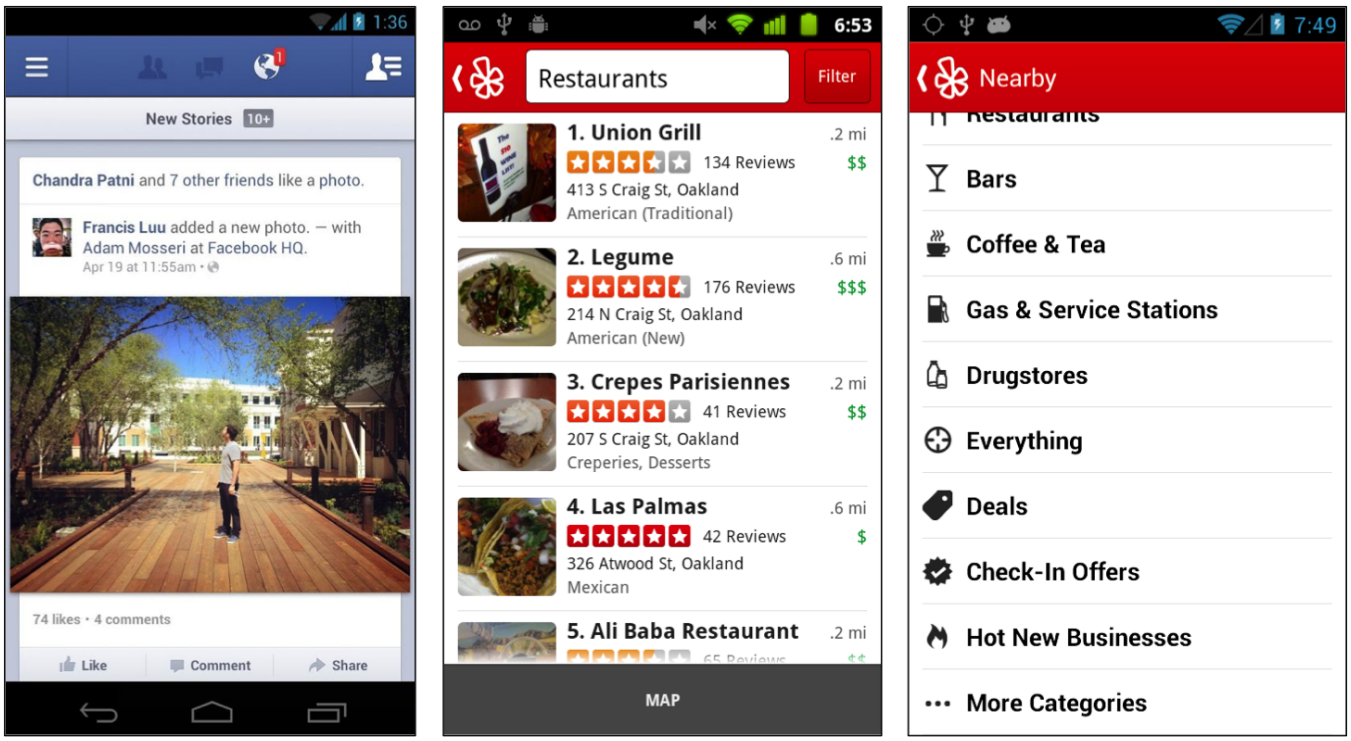


Figure 12: Layouts that can hold a dynamic number of child elements, such as lists, present another traversal challenge. First, the state of the application changes depending on the number of items loaded. A severe case is the *Infinite Scroll* problem, signifying when a content layout dynamically keeps adding content, such as Facebook and Twitter feeds. Second, each item in the list may have a different callback handler. A traversal algorithm, especially a time-limited one, cannot interact with every item in the list.

two ways. First, because the number of items can change dynamically, a change in the list item changes the app state. A severe case of this is the *Infinite Scroll* problem, where the application keeps adding content as the user scrolls down, such as Facebook and Twitter feeds. Second, as items in a list may register different callback handlers, the traversal cannot simply ignore them. However, the traversal algorithm also cannot interact with every element, especially if the traversal is time-limited. Lee *et al.* proposed a solution, AMC, to this problem by checking the callback handlers and only clicking on the first two items if callback handlers are homogenous [92]. There exist special cases where a long list of items is followed by an item with a different callback handler, e.g., the Yelp category list in Figure 12.

4.3.5 Non-Deterministic App Behavior

Some apps exhibit non-deterministic behavior, for instance, through presenting the user with random pop-ups to rate the app on Google Play, requesting the user to share the app with friends, or sporadically asking the user to login. While these behaviors seem innocuous at first, they can be detrimental to app traversals, especially time-limited ones. Dialogs that guide the user to the Google Play store or to other external apps

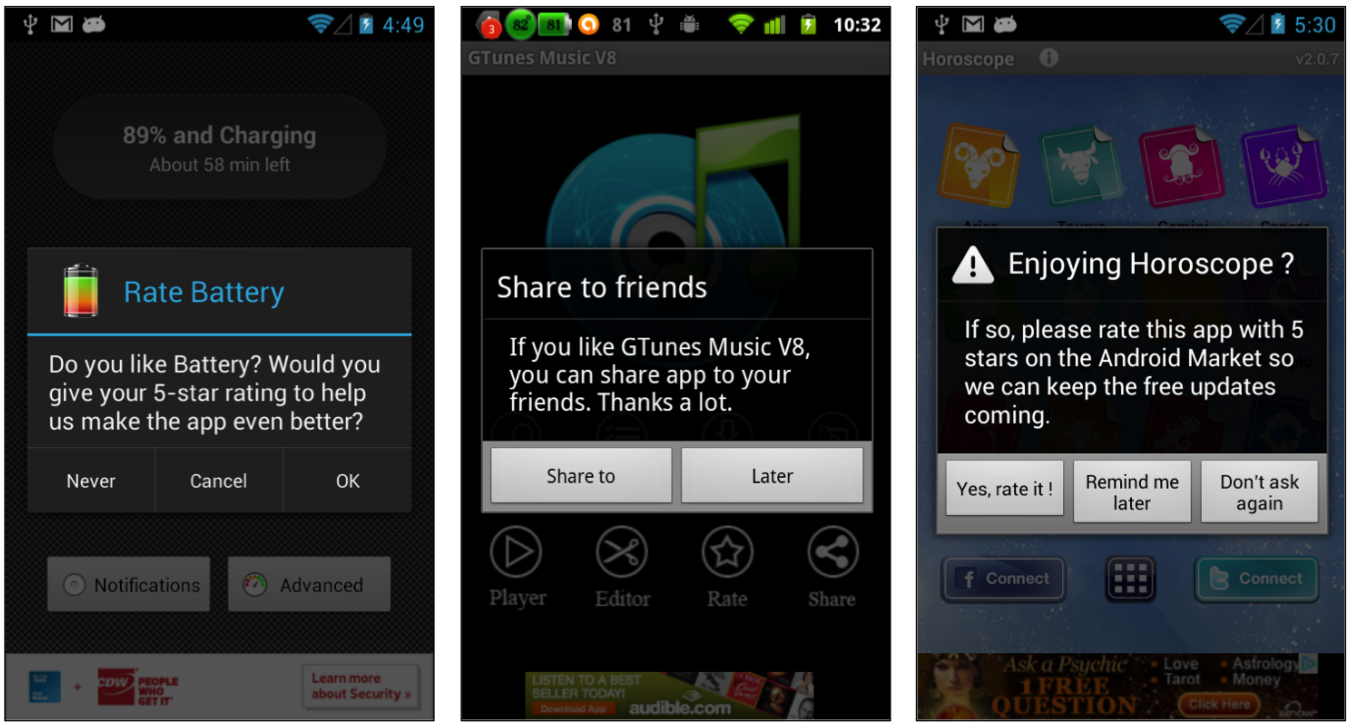


Figure 13: Apps may implement non-deterministic behavior. Such implementations make traversals non-deterministic as well, i.e., two traversals on the same app may give very different results. This figure presents several examples of app behavior that is triggered only sporadically.

cause the traversal algorithm to exit the target app, which costs valuable traversal time. Sporadic optional login screens present an especially difficult case, as these screens are generally part of the app and a naïve traversal algorithm would not be able to work around them. Figure 13 presents several non-deterministic app behaviors.

4.3.6 Special (Physical) Input

A class of apps requires special forms of input to enable features. For instance, a number of apps are tied to physical trackers (e.g., Fitbit[15]) or other peripherals such as the Samsung Galaxy Gear watches[16] or the Square card readers[17]. It is non-trivial to emulate the behavior of these peripheral devices. Traversing such apps may require human intervention and physical devices, or could perhaps be supported through software emulated versions of these devices. Figure 14 presents several apps that require special forms of (physical) input.

[15] `http://www.fitbit.com`

[16] `http://www.samsung.com/uk/consumer/mobile-devices/galaxy-gear/galaxy-gear/SM-V7000ZKABTU`

[17] `https://squareup.com`

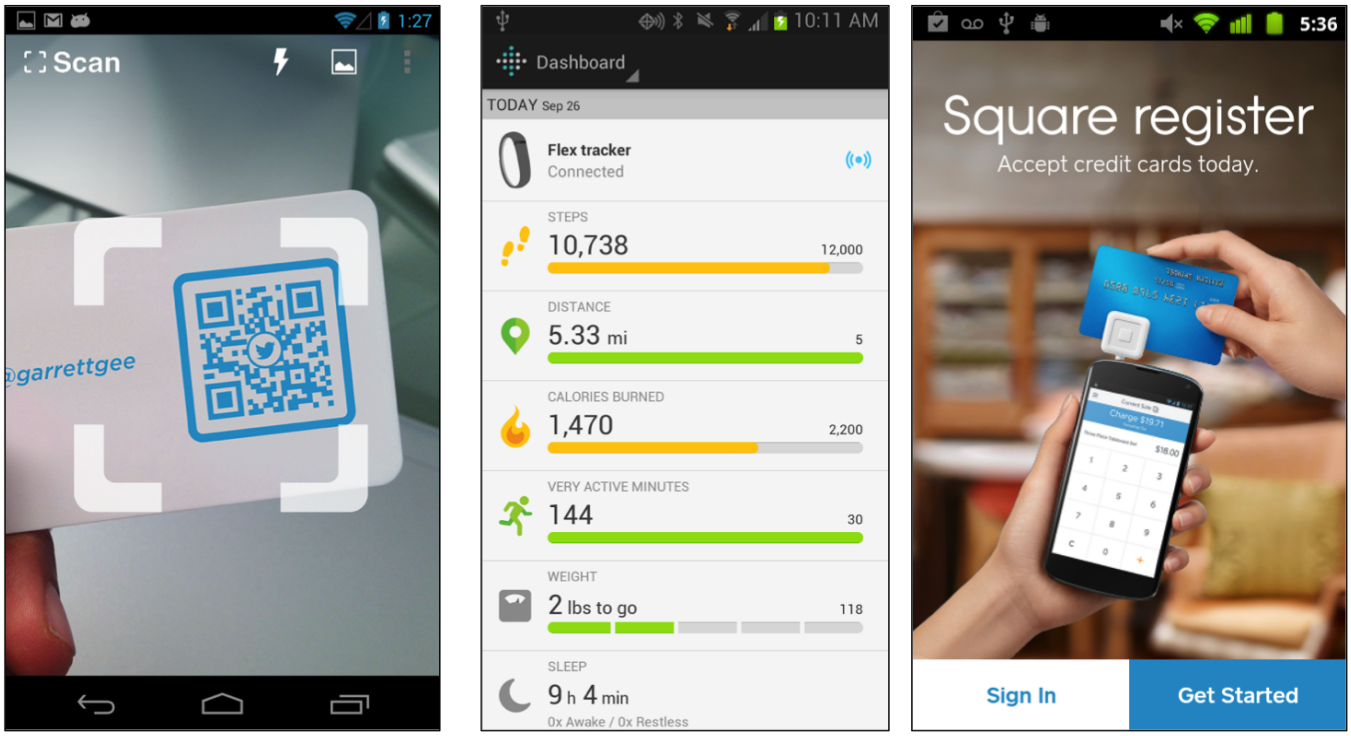


Figure 14: Some apps require special forms of input to trigger app behavior. This figure presents several such examples (right to left): QR Code Reader, Fitbit, and Square. Traversal of these apps may require human intervention or software to emulate the input.

4.3.7 Special Setup

Apps may require initial setup prior to being used. A simple example of special setup is requiring user authentication. However, other forms of setup may complicate app traversals. For instance, the LinkedIn Pulse app is a news app that asks the users to input their topic or areas of interest when it first launches. It is common for apps to also have a settings page to enable or disable features. While these features are great in giving users freedom, they may offer too many alternatives for a traversal algorithm. In severe cases, the availability of many options may lead to state explosion and confuse the traversal algorithm [33]. Figure 15 presents two applications that will present drastically different content based on the user setup.

4.3.8 System Apps

System apps allow the user to change system settings or behavior. For instance, the Advanced Task Killer app allows users to kill other applications and processes. Battery Booster Lite allows users to change system settings such as enable or disable WiFi or Bluetooth or turn on Airplane mode. Refer to Figure 16 for screenshots. Traversing these types of apps is challenging as the system behavior and functionality

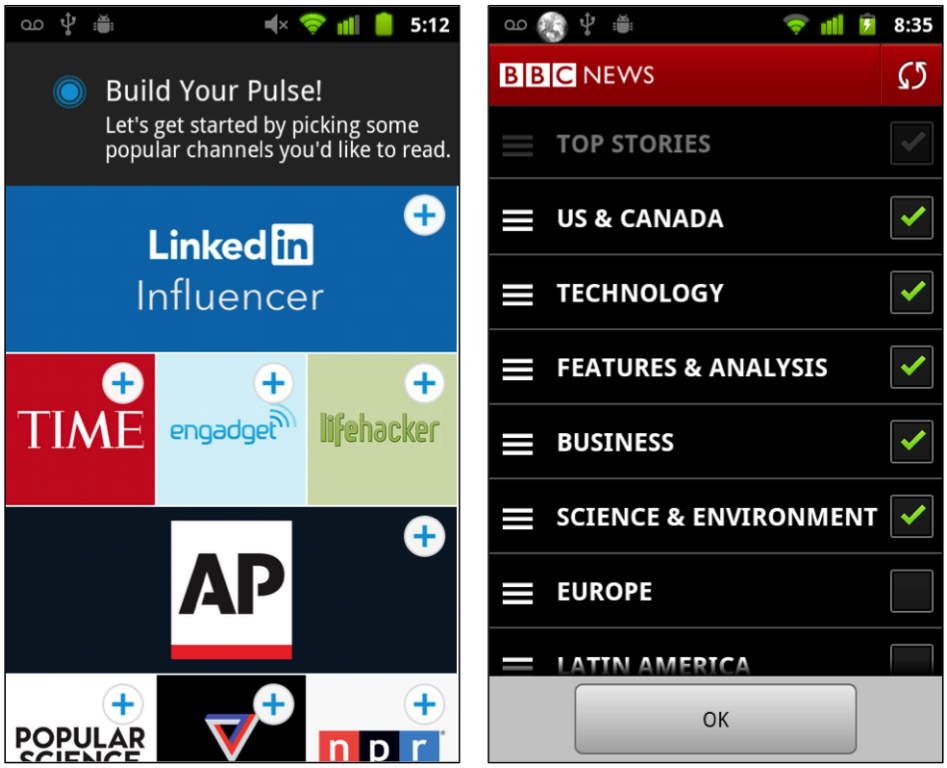


Figure 15: The LinkedIn Pulse app requires that users select their topics of interest during the first launch. The BBC News app offers users different content based on their selections. Traversing applications with manual setup procedures may result in entirely different application behavior.

may change while the traversal environment and result in reduction in traversal coverage and unexpected results. For instance, turning off the WiFi would likely change the app behavior for Battery Booster Lite as it would no longer be able to access the Internet.

4.3.9 System Instability

Squiddy traversals were all conducted on a physical phone instead of an emulator to simulate a realistic app environment. The Android system is generally stable. However, during traversals, it was common for phone processes to crash on the TaintDroid patched Android build. It was also common for the Android Debug Bridge to lose connection [62]. While testing Squiddy v1, many traversals would terminate early as a result of processes crashing on the phone and adb connection issues. As Squiddy uses TaintDroid to monitor privacy leaks, there is not much of a work-around for this problem. The second version of Squiddy makes heavy use of software tolerance techniques such as retries and recovery blocks [96] to mitigate system instability problems.

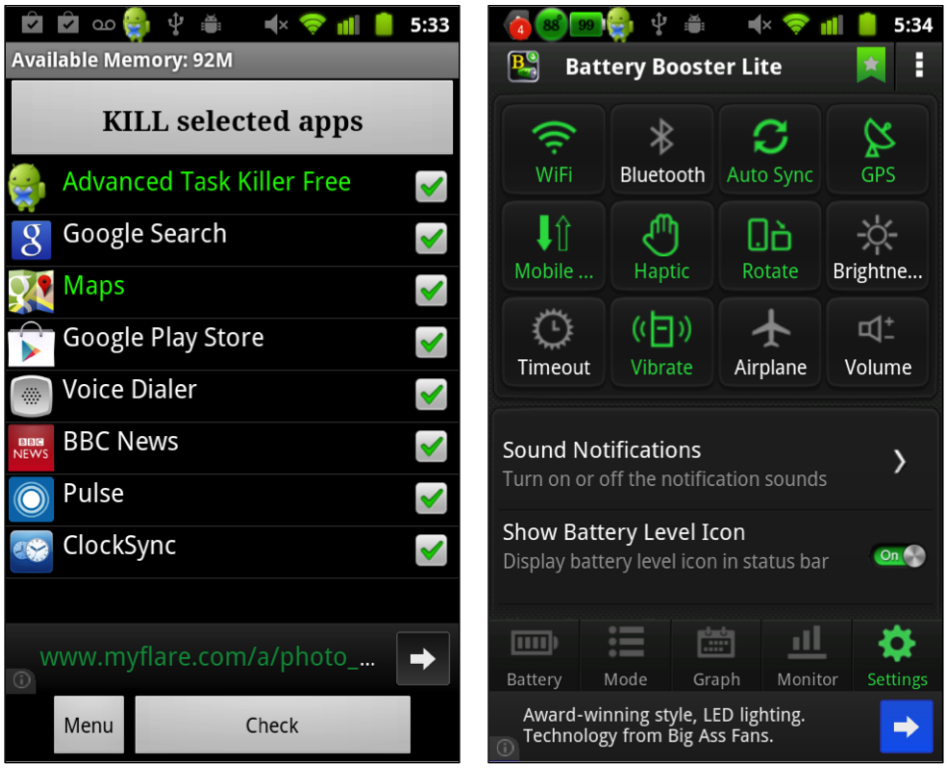


Figure 16: System apps can alter the state of the underlying Android system and its settings. Automated traversal of such apps is difficult as the system environment may change as a result of the traversal interactions. For instance, Advanced Task Killer can kill other apps or the Battery Booster Lite app can toggle WiFi state.

4.3.10 Time synchronization

Squiddy monitors privacy leaks using TaintDroid. TaintDroid logs are timestamped using the Android device's clock. However, interaction instructions are sent from a host PC running the traversal algorithm. As such, interactions have the timestamp of the host PC's clock. In order to have a correct mapping of sensitive transmissions to application states, it is important for the Android device clock and the host PC clock to be synchronized. While traversing applications with Squiddy v1, there were times when the privacy leak detected through TaintDroid would not match the state during which the privacy leak occurred. In the experimental setup, the Android device used did not have a cellular connection available. As a result, the device time would drift and cause inconsistencies in Taint Log detection time.

4.3.11 User Authentication

Many apps are tied to service providers that require user authentication, e.g., Facebook, Twitter, and Pinterest. Unfortunately traversal algorithms cannot go beyond the welcome screen because of user au-

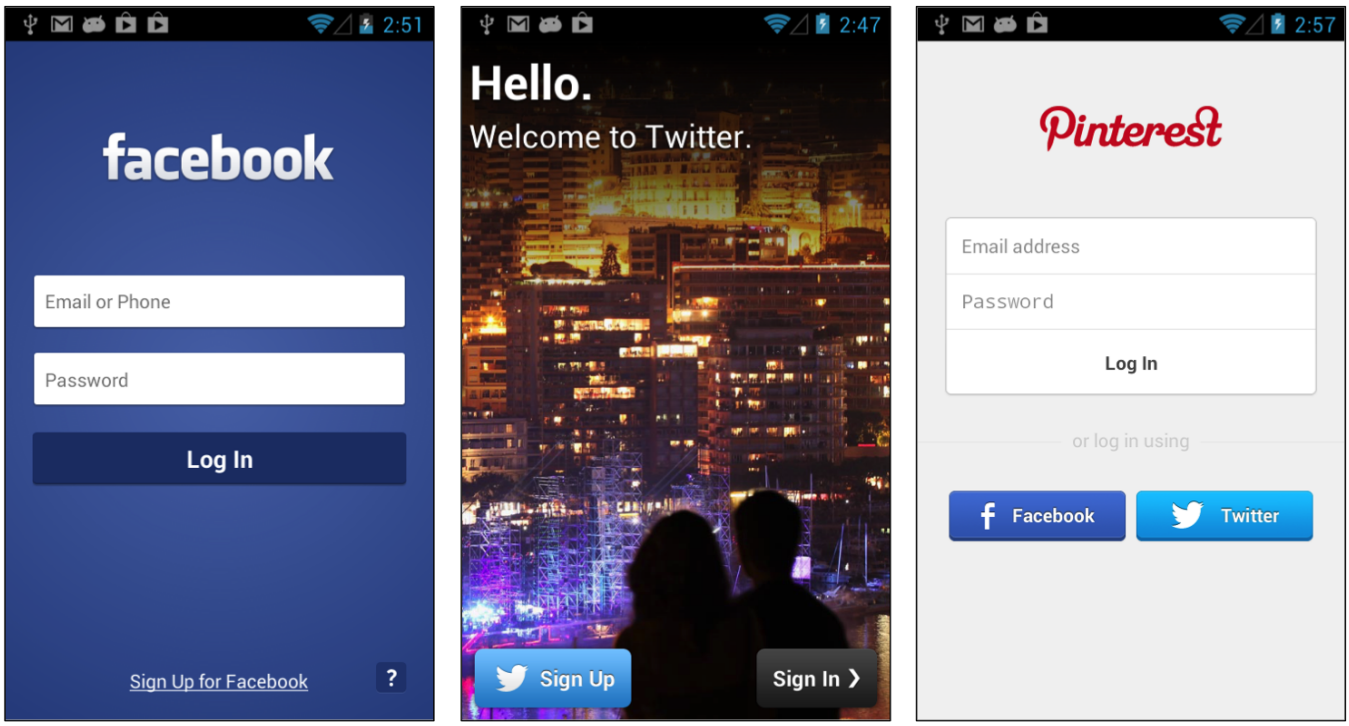


Figure 17: The traversal algorithm may not be able to go beyond the welcome screens for apps that are tied to service requiring user authentication.

thentication. One approach is for privacy analysts to provide user authentication at the time of traversal. However, this approach does not work well in the case of automated analysis engines. For many services, account creation is guarded by CAPTCHAs that are designed specifically to prevent automatic account creation [2]. As such, traversal algorithms cannot automatically generate user credentials. Traversal algorithms may try to use crowdsourcing to their advantage to ask crowd participants to create logins for the traversal and automatically verify and use the credentials at traversal time. Nevertheless, dummy accounts would most likely not be populated with user data, which would make coverage using dummy accounts partial. Some applications may provide partial features even without a login. For instance, the Yelp app makes login optional and provides many key features available without authentication, e.g., search, reviews, directions. Another option would be to call app activities and receivers that were discovered using static analysis techniques using guess inputs. Without cooperation from the service end-point, the gains from such workarounds would be limited.

4.3.12 Virtual Keyboard

The Android virtual keyboard allows users to type on their screen. When an input text area is selected, the virtual keyboard is presented to the user. However, during a traversal, if the keyboard is visible, the traversal algorithm can no longer interact with the UI components underneath the keyboard. As such,

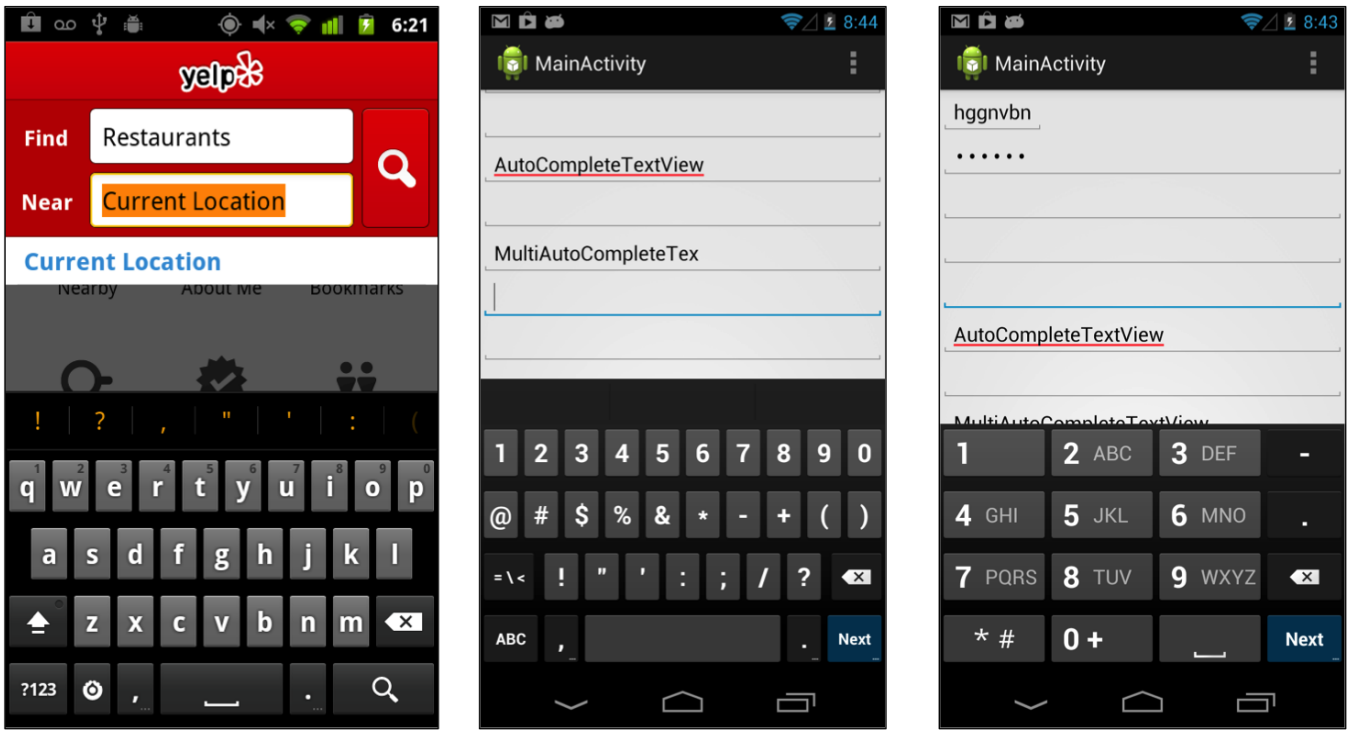


Figure 18: The traversal algorithm has to intelligently close the virtual keyboard to access UI components hidden by the keyboard. The Android keyboard looks different on different devices and different versions of Android. A variety of keyboard types are also supported to enter different types of data, such as phone numbers, email addresses, or standard text. The left image shows the virtual keyboard on Android 2.3.4 on a Nexus S. The middle and right images show two different types of virtual keyboards on the Galaxy Nexus. Considering the wide variety of virtual keyboards, simple image comparison methods cannot be used to detect the presence of keyboard on different devices.

the traversal algorithm has to intelligently close the keyboard to access the UI elements below. A simple method to close the keyboard is to send a 'back' button press command to the device. However, the challenge is in knowing when the keyboard is visible. The Android system does not expose the presence of the virtual keyboard externally. At first glance, this may be an easy problem to solve using a simple computer vision algorithm to check for the existence of the keyboard. However, this approach does not work for the variety of devices and brands as virtual keyboards do not follow the same design. Furthermore, different devices have different screen resolutions which result in different size keyboards.

Another challenge is that the virtual keyboard changes the input range based on the type of input box selected. For instance, the keyboard presented for entering phone numbers is different than the one for entering standard text. As such, even if an image comparison was made to check for the presence of a virtual keyboard, the comparison would have to be made against the variety of different virtual keyboards. Figure 18 shows keyboard layouts for different versions of the Android OS, Android devices, and types of input.

4.4 Second Iteration of Squiddy

The second iteration of Squiddy makes drastic improvements on the preliminary version. This section presents how Squiddy v2 mitigates some of the traversal hurdles and then delves into the details of how the second version of the traversal algorithm works. Squiddy v2 does not tackle all the previously mentioned challenges as the main goal for the thesis is to offer privacy analysts an interactive tool to evaluate mobile app privacy, which does not rely solely on the traversal algorithm. Addressing all the traversal hurdles is a challenging problem in itself and would deserve a separate thesis. However, Squiddy v2 offers solid improvements and provides reasonable app coverage to enable in-depth analysis with the interactive GUI.

4.4.1 Mitigating Activity Stuffing through State Discovery

As Android activity components can support more than one UI layout by dynamically changing the layout during run-time or using UI elements such as Tab views, it is important to distinguish different layouts and components of the activity. Squiddy v2 adopts a similar approach to that of Lee *et al.*'s traversal algorithm, AMC [92]. Specifically, Squiddy determines different states of an activity by computing a hash of an activity every time that it visits a new activity or interacts with an activity.

Similar to the first version of Squiddy, Squiddy v2 reads and interacts with app UIs using TEMA [122]. Reading the app UI presents Squiddy with a hierarchy of the app UI elements, i.e. the UI DOM tree. Squiddy computes a hash of the UI structure, referred to as the *structure hash*, in a similar way to AMC. Specifically, the DOM is traversed in depth-first order, hashing the tag name, level, and sibling order of each XML node, and omitting all other content. The hash is then used as a unique identifier for each state. The only difference between the hash computed by AMC and the one computed by Squiddy is that Squiddy also uses the activity name in the structure hash. This is in case more than one activity in the app uses the same layout. The two activities could be presenting different content and handling the content differently.

4.4.2 Summarizing UI Lists

Lists that may dynamically change during run-time present a challenge in computing the structure hash. Specifically, if the number of the items on the list changes, the structure hash would also change. This scenario would be fairly common in apps that present search results, instant messages, or emails. In this case, Squiddy also adopts a similar approach to AMC. When computing the structure hash, Squiddy only uses the first two elements in the list to compute the hash for the list. The remaining items in the list can still be interacted with, but are not used in determining the hash value.

As app developers can create their own lists and multi-element views by extending Android classes, it is important to know which classes provide these types of views. TEMA only presents the leaf class names for element types and does not provide information about parent classes that may have been extended [122]. To distinguish these classes, Gort exposes the callback handlers for list item handlers in the DOM tree that TEMA sees, similar to how an element's *x, y* coordinates are presented to TEMA. Specifically, I modified the code under frameworks/base/core/java/android/View.java and frameworks/base/core/java/android/widget/AdapterView.java.

View.java is modified to expose the following callback handlers:

- mOnClickListener
- mOnLongClickListener
- mOnTouchListener
- mOnCreateContextMenuListener

AdapterView is modified to expose the following callback handlers:

- mOnItemClickListener
- mOnItemLongClickListener

Note that there are other handlers that could be modified if a traversal algorithm were to use more complex interactions, such as swipes. However, for the purposes of Squiddy, only click interactions and keyboard key presses are used.

4.4.3 Interacting with Non-Deterministic Dialogs

As previously noted, apps can exhibit non-deterministic behavior, such as sporadically presenting the user with dialogs to rate the app, or requesting the user to share the app with friends. Such behaviors make the outputs of the associated app traversals non-deterministic as well. A traversal that arrives at a non-deterministic dialog could take a different route through the app than one that does not.

Squiddy implements logic to distinguish when a dialog is present. Then, through using heuristics based on design patterns, Squiddy attempts to dismiss the dialog. The first approach with any dialog is to try to dismiss it. If the dialog cannot be dismissed by emulating a *back* button press, Squiddy looks for buttons with labels associated with dismissing interactions, such as, *cancel*, *no*, *not now*, *later*, *don't ask*, etc. If Squiddy cannot find such buttons, it looks for buttons with positive labels, such as, *okay*, *ok*, *agree*, *continue*, and *accept*. The use of positive buttons is important as it allows the traversal algorithm to jump through consent forms, terms and services, and End-user License Agreements (EULAs).

Note that while the above approach is useful in the case of sporadic or random dialogs, it does not disturb the traversal of deterministic dialogs. The Squiddy algorithm is designed to interact with all UI elements that it sees. As such, if it arrives at a screen or dialog that contains elements that have not been explored before, it gives priority to interacting with those elements. Therefore, all deterministic UI elements are exercised during the traversal.

4.4.4 Time Synchronization

There exist two simple work-arounds for the time synchronization problem. One option is to use an emulator running on the host PC for traversals. However, using an emulator would deviate from simulating a realistic mobile device environment. Another option is to use a physical device with a cellular connection that automatically synchronizes the device time, and a host PC that automatically synchronizes time. The assumption here is that both devices are connected through USB and located in the same time zone. A more difficult alternative is to use a rooted Android device for the traversals. Rooted Android devices support setting the device time, which can then be used to synchronize time with the host PC.

Squiddy v2 addresses the time synchronization problem by calculating a time offset for the host PC and the Android device. Specifically, Squiddy determines the time offset by reading the Android device time and uses the offset to adjust its timestamps for its interaction commands. In practice, this works well, especially as there is a sleep period after each interaction. However, having precise time synchronization [99, 113] would be valuable to correctly map sensitive behavior while the app is changing from one state to another, i.e., while app state is transient.

4.4.5 Detecting the Virtual Keyboard

Detecting the presence of a virtual keyboard is crucial as the traversal algorithm cannot interact with UI elements that are hidden by the virtual keyboard. Squiddy v1 addressed this problem by comparing screenshots of the apps at various stages with a known image of the Android keyboard. While this approach works on a single device or devices with similar type keyboards, it does not scale when traversing on a variety of devices. Furthermore, because the keyboard layout may be different based on the type of input required (e.g., phone number vs. email address), the image comparison would have to compare each screenshot with a variety of keyboard layouts. Also, based on app implementation, the keyboard layout may change to match the orientation of the screen, i.e., the keyboard can be presented in portrait or landscape format. As such, the algorithm would have to check against different orientations as well.

Squiddy v2 takes two approaches to detect the presence of the keyboard. The first approach is one based on computer vision (specifically, optical character recognition (OCR) and a sequence alignment

algorithm). It should work reasonably well on any Android device without modifications to the Android build running on the device. The second approach modifies the Android OS to expose the presence of a virtual keyboard as an external variable, accessible through adb [62]. The kernel modification approach is more reliable. As such, Squiddy v2 relies on the kernel approach first, and if not successful (i.e., cannot find the variable), falls back on the OCR plus sequence alignment.

Sequence Alignment: The sequence alignment approach works based on the assumption that the keyboard layout does not change, e.g., the order of keys is always the same on a QWERTY keyboard. The algorithm works in the following way. After visiting each new activity or interaction, Squiddy takes a screenshot of the app. Squiddy then uses OCR to detect the letters present on the screen. The current implementation uses the tesseract OCR engine [80]. Squiddy also performs OCR on different orientation of the screen to get the best matching of the characters. At this time a naive approach would check against a specific keyboard sequence, e.g., *qwertyuiop*. However, OCR is not perfect, as such, not all letters on the virtual keyboard would be detected. Using a naive approach would lead to a false negative in cases where OCR does not correctly match each character on the keyboard. Based on experience with tesseract, it was common for characters to be skipped or identified incorrectly. To address the recognition performance problem, Squiddy uses the Needleman-Wunsch global optimum sequence alignment algorithm to match a keyboard sequence [103].

Using the sequence alignment algorithm provides a similarity score for each line recognized by the OCR against potential lines in a keyboard layout. The keyboard recognition algorithm runs on layouts available by the Android virtual keyboard, e.g., standard text, phone numbers, and email. Lines read are matched against different keyboard layout templates separately. Finally, the score computed is thresholded to a binary result on whether the virtual keyboard is present on the screen or not.

The sequence alignment approach is not without limitations. For instance, for apps that use a different keyboard layout (e.g., Chinese Traditional layout), the keyboard layout has to be introduced to Squiddy. Traversals should be performed using a standard QWERTY virtual keyboard rather than other keyboard layouts, e.g., Dvorak. Otherwise, the Dvorak layout should also be introduced to the algorithm.

Android OS Modification: Android makes a number of system properties externally available through adb [62], for instance, information about the build running on the device or the device model and brand. Another approach for determining virtual keyboard presence is to modify the Android OS to expose an external variable that can be checked by the host PC. Following this approach, I modified the Android OS to expose a variable to show when the virtual keyboard is visible. Specifically, I edited the frameworks/base/core/java/android/inputmethodservice/InputMethodService.java file to add system properties for the hideSoftInput and showSoftInput functions. Furthermore, I gave the keyboard process permission to write system properties in a configuration file, android_filesystem_config.h.

This method of making the presence of the virtual keyboard to the host PC is the most reliable option exercised by Squiddy. Specifically, it does not require any knowledge of the keyboard layout, OCR, image detection, or sequence alignment. However, the method comes at the cost of modifying the Android OS. Specifically, any traversal algorithm that solely relies on this method would have to run on a modified Android platform regardless of whether the algorithm runs on an emulator or physical phone. The Squiddy implementation falls back on the sequence alignment approach, as such, it is not limited to certain types of devices. However, considering that Gort requires TaintDroid to monitor privacy leaks, and that TaintDroid is a modified version of Android, the minor changes to the files above do not necessitate a major modification on top of what is already required.

4.4.6 Squiddy v2 Algorithm Details

Squiddy v2 improves on v1 to provide better coverage of apps. Using Squiddy v1, traversing apps involved many communication disconnects with Android and also problems reading the GUI interface. As such, I enhanced the TEMA framework [122] with retries and recovery blocks [96]. Squiddy v2 also uses a *depth-first search* (DFS) algorithm instead of the original *breadth-first search* approach. This design allows Squiddy to delve deeper into the app without having to jump back to previous states. Moreover, Squiddy v2 borrows techniques from Lee *et al.*'s AMC traversal algorithm to explore and discover app states rather than activities [92]. Hence, it produces a *State Flow Graph* (SFG) rather than the original Activity Flow Graph. One activity can have many states, as such, an SFG is more comprehensive. Squiddy v2 also uses static analysis techniques to determine and explore only activities that belong to the app.

Squiddy v2 considers each distinct presentation of an app GUI to be a state of the application. Similar to AMC, Squiddy v2 keeps an exploration graph of the app. The algorithm starts with the main screen of an app. For each element that Squiddy can interact with, it adds an unexplored edge to the exploration graph. When an edge is explored, Squiddy marks the edge as explored and chooses another unexplored edge to attempt. Priority is given to unexplored edges that can be reached using the fewest number of transitions. Squiddy chooses an edge for exploration using a breadth-first approach on the exploration graph. Specifically, it gives priority to edges that can be reached on the current screen.

In order to explore an edge, Squiddy uses TEMA to interact with the element associated with the edge. Upon interaction, there can be three different outcomes. One, the app state does not change based on the interaction. Two, the app transitions to a state that it has already seen. In this case, Squiddy adds an edge from the starting state to the new state based on the recently explored edge. Third, the app transitions to a state that it has not seen before. In this case, a new state is added to the exploration graph using the recently explored edge. Squiddy takes a screenshot of the new state and associates it with the state.

Similar to Lee *et al.*'s work, Squiddy v2 identifies states based on a structure hash of the GUI. For details, refer to §4.4.1. The algorithm also deals with non-deterministic transitions as AMC does, by removing non-deterministic edges from the exploration graph. However, Squiddy tries to also identify some screen features during the traversal, namely, dialog and keyboard presence. Through this identification, Squiddy's domain knowledge allows it to work around dialogs, especially dialogs with non-deterministic behavior (§4.4.3), and discovers and interacts with components hidden by the keyboard (§4.4.5). For all other purposes, the algorithm works similar to AMC [92], however, Squiddy does not implement multi-sequence interactions, as the coverage provided using single interactions is sufficient for its purposes.

4.4.7 Squiddy v2 App Coverage

Squiddy v2 explores and discovers app states rather than activities. In this way, the algorithm works around the activity stuffing problem. The states in the app are detected as a different hash value based on the layout. Similar to Squiddy v1, the duration for traversals is limited to 60 minutes. Traversing beyond this amount of time does not scale and results in severely diminishing returns. Furthermore, a majority of apps that tested finished traversing in under 30 minutes. Table 3 presents the app coverage results for Squiddy v2.

While Squiddy v2 provided much better coverage of apps (μ=49.5%, σ=20.1 percentage points) than v1, it still has room for improvement. More specifically, it was noted that the majority of the cases, where states were not explored further, required a sequence of interactions, e.g., a swipe followed by a click. By extending the algorithm to perform sequences of interactions, app coverage would increase [92]. There were also cases where an app would require login for additional features, for example, Yelp requires user login to show its friends feature. Nevertheless, the focus for this thesis is not on traversal algorithms, rather to offer an interactive tool for privacy analysts to evaluate mobile app analysis and to study how analysts use such tools. As such, the development of Squiddy v2 continued until it offered a reasonable amount of coverage to provide reasonable input for the heuristics, crowd analysis stage, and the GUI tool.

The next chapters present how Gort uses the results from app traversals to run heuristics to find potential app privacy problems, to determine the legitimacy of privacy leaks by relying on crowd input, and also to present app behavior using an interactive GUI tool for privacy analysts.

App	Squiddy v2 Discovered	Manually Discovered	Squiddy v2 Coverage (%)	Squiddy v1† Coverage (%)	Time (s)
Best Parking	8	14	57.1	7.1	284
Bible App	15	40	37.5	5.3	3600
CityShop	13	23	56.5	5.9	1274
ColorNote Notepad	6	37	16.2	5.0	470
EBay	11	21	52.4	2.3	1162
GasBuddy	20	24	83.3	9.5	3600
G. Finance	6	14	42.8	12.5	1426
G. News & Weather	4	11	36.3	16.7	858
G. Search	5	6	83.3	12.5	794
Groupon	15	23	65.2	4.3	3600
KBB.com	13	24	54.2	44.4	2806
QR Code Reader	10	11	90.1	25.0	1113
Relax Timer	5	11	45.4	16.7	3600
Stopwatch & Timer	4	10	40.0	11.1	565
Tag	3	6	50.0	14.3	44
Tumblr	7	17	41.1	6.7	3600
Weather Channel	3	18	16.7	6.3	160
Wikipedia	3	11	27.3	50.0	138
Yelp	22	49	44.8	9.0	3600

Table 3: This table presents percent app coverage using the Squiddy v2 traversal algorithm. Coverage is calculated as the number of states discovered by Squiddy divided by the number of states discovered by manually interacting with the application. The mean and standard deviation for coverage were 49.5% and 20.1 percentage points, respectively. Applications that took 3600s to traverse reached the traversal timeout (60 min). †The *Squiddy v1 Coverage* column is provided as a point of comparison. The coverage estimate for Squiddy v1 is a preliminary estimate as it is based on activities and not states.

5 Heuristics to Identify Potential App Privacy Problems

After completion of the first implementation (v1) of Gort, it was observed that many simple questions about app privacy required analysts to dig into network request details or app permissions. To optimize for analysts' time and attention and also to minimize the tedious and repetitive nature of analysis tasks, I began exploring the idea of supporting a set of heuristics to help analysts find app privacy issues. One example heuristic is as follows: *If an app has access to the device's location and also the Internet, the app can potentially use its access privileges to track users.* Using heuristics, analysts do not have to scrutinize the details of the app behavior and network requests to find potential privacy problems.

I compiled a set of heuristics for mobile app privacy in three steps. First, I distilled a list of heuristics commonly used by researchers and privacy analysts to evaluate app privacy and security [45, 78, 81, 126]. Second, I manually inspected over 40 apps to gain insight into Android app behavior and devised heuristics to detect potentially privacy-intrusive conduct. Third, I elicited heuristics by interviewing 12 experts. I presented experts with mobile apps and the associated app descriptions, screenshots, and sensitive transmissions and asked them to analyze the privacy related behaviors of the apps. Using the above approaches, I compiled over 100 heuristics for app privacy analysis. The goal of compiling heuristics was not to discover all heuristics targeting app privacy, but to provide a framework that helps others organize, think about, and come up with other heuristics in a structured manner.

5.1 Interviews with Experts for Eliciting Heuristics

I recruited 12 participants for interviews, consisting of 9 security and usable security researchers, 2 developers, and 1 IT professional. 3 of the participants were female and the other 9 were male. 10 out of the 12 participants were between 20-29 years old. Recruitment focused specifically on people who had a background in security, development, and IT and prior experience using smartphones. Participants were compensated with $30 Amazon gift cards. Each participant was presented with two apps from a set of apps (Yelp, Fox News, Pulse News, Horoscope, TossIt, ColorFind) for analysis, one app at a time. Information about an app was incrementally presented to the participant to implicitly elicit heuristics based on how the participant analyzed the app. After participants studied the two apps, they were explicitly asked for other potential heuristics.

Participants were provided app information incrementally to gauge how they used the information. Specifically, first, they were presented with the app description, screenshots, and permissions as available from the Google Play store. They were then asked to analyze the app based on the information presented to them. Next, participants were provided with the network transmissions of the app including the IP address, host name, server registrants, and the type of sensitive information sent (e.g., location, phone

number). They were asked to analyze the app again based on the new information. Next, participants were provided with the screenshots associated with the transmissions and asked to analyze the app again. They were also asked to speculate why the app would transmit the sensitive information and whether they deemed it necessary. Finally, they were asked to determine a privacy score for the app and share how comfortable they felt with the score. Using the above approach, in addition to heuristics from prior work and based on our own inspections, over 100 heuristics were compiled.

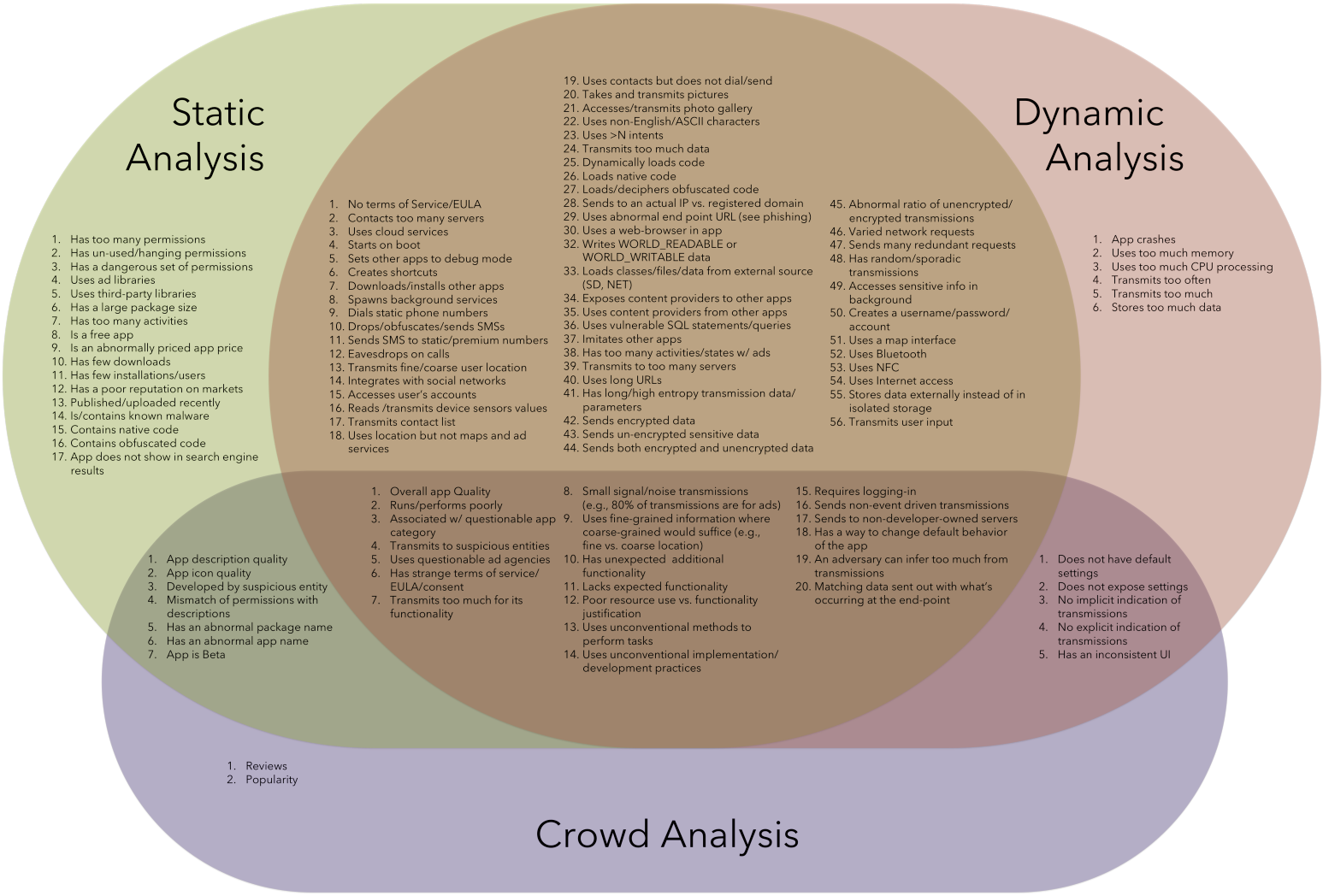


Figure 19: The first iteration of consolidating the heuristics focused on how heuristics could be divided based on the technique(s) necessary to determine their output. A combination of static, dynamic, and crowd analysis could be used to determine the output of heuristics. Heuristics belonging to the Crowd Analysis set would require human judgment and inference in addition to traditional computation-based techniques. For a larger size diagram, refer to Appendix B.

5.2 Heuristic Consolidation Results

Over 100 heuristics were obtained. Figure 19 presents the obtained heuristics in the form of a Venn diagram based on how the heuristics' outputs can be determined. The Venn diagram was the first iteration of categorizing the heuristics. The purpose of the Venn diagram is to indicate the techniques that a developer can use to implement the heuristics. The mapping of heuristics to specific techniques is based on the Android platform, but, should be extendable, perhaps with minor revisions, to other mobile platforms. For a larger version of the diagram, refer to Appendix B.

5.2.1 Taxonomy of Heuristics Based on App Components

Following the Venn diagram, I consolidated the heuristics into categories with three other researchers and one developer, none of whom were involved in the project. Based on this consolidation, I came up with a taxonomy for heuristics. The consolidation led to three top level categories, which generally describe apps based on their components:

Presentation: Presentation focuses on the surface quality of the app, consisting of the information available on Google Play and the general functionality and presentation of the app. Presentation heuristics do not require in-depth analysis of the app's implementation (e.g., source code, binary) or transmissions.

Communication: Communication deals with app network transmissions, specifically focusing on what is sent, to whom, when, and with what accuracy.

Implementation: Implementation focuses on app source code and binaries.

Each category is further broken down into subcategories to further illuminate the types of heuristics that would belong to the category. Table 4 presents the major heuristic categories and their associated subcategories. An example is provided for each subcategory of the taxonomy.

5.2.2 Taking into Account App Context and Association

Another heuristic categorization approach that emerged from the consolidation process was based on aspects of the app ecosystem. While a categorization based on app components provides a micro-level view of the heuristics, apps are often analyzed in relation to other apps and the entities with which they associate. Such analysis generally focuses on the contexts and associations of apps, which provide a macro-level perspective. For instance, while detecting sensitive resource use could be a heuristic, the resource use may be acceptable to end-users based on the context of the app. The *context* and *association* categories are defined as follows:

Presentation

1. Poor Appearance Quality (e.g., poor app description)
2. Poor Overall Quality (e.g., inconsistent UI)
3. Lack of Functionality (i.e., does not deliver expected features)
4. Unexpected Functionality (e.g., Blackjack app finds friends)
5. Rare Functionality (e.g., requires login)
6. Misalignment (e.g., use of permissions not justified by features)
7. Unconventional Behavior (e.g., requests user to share app)
8. Lack of Transparency (e.g., no Terms of Service)
9. Misrepresentation (e.g., fake app for a well-known service)
10. Poor Reputation (e.g., developer's other apps have low ratings)
11. Un-trusted Source (e.g., from app market with lots of malware [98])

Communication

1. Unwarranted Data (e.g., Horoscope transmits users' location)
2. Unwarranted Frequency (e.g., Horoscope transmits users' location on every screen)
3. Unwarranted Resolution (e.g., restaurant finders asks for exact rather than approximate location)
4. Improper Protocol (e.g., does not encrypt sensitive information)
5. Unexpected Receivers (e.g., sends user's location to ad networks)
6. Poor Implementation (e.g., sends sensitive data in the clear)
7. Poor Architecture (e.g., no support for disconnected operation)

Implementation

1. Unwarranted Resource Use (e.g., Horoscope requests location)
2. Improper Metadata (e.g., old version, MD5 checksum mismatch)
3. Poor Implementation (e.g., stores encryption keys on SD card)
4. Poor Architecture (e.g., uses database for temporary files)
5. Unconventional Implementation (e.g., implements crypto)
6. Malicious Implementation (e.g., deletes users' data)
7. Non-reputable Components (e.g., use of strange crypto libraries)
8. Advertisement Components (e.g., AdMob)
9. Analytics Components (e.g., Flurry [18])

Table 4: I elicited heuristics to find potential app privacy problems by conducting interviews with experts. This table presents a taxonomy of the heuristics types and subtypes, which generally describe fundamental app components. For each subcategory, the table presents an example heuristic.

Context: An app's context is defined as the purpose and claims of the app, as well as where the app belongs in relation to the other apps in the app ecosystem.

Association: Association refers to the entities and individuals related to the app, for instance, the developer, advertisement networks, analytics agencies, and third-party libraries.

While app component categorization answers the "what components of the app should be analyzed?" question, the context category focuses on "how the components should be analyzed?" The association category frames the app analysis with respect to "who or whom should be considered as part of the analysis?" Figure 20 presents the heuristics associated with the context and association of apps.

5.2.3 Human Judgment and Expertise

Determining the results of some heuristics may require human judgment or inference in addition to traditional static and dynamic analysis approaches. Figure 19 presents such heuristics grouped under the *Crowd Analysis* set. As an example, using only traditional computational techniques, it may be difficult to determine that an app has a strange Terms of Service or EULA, or that an app has poor overall quality. Additional heuristics may be created and adopted to help with such heuristics. For instance, one could use machine learning and natural language processing techniques to rate the quality of the Terms of Service or EULA for an app. In the case of overall quality, one could come up with additional heuristics to guess the overall quality of an app, e.g., color scheme consistency, font readability, UI usability. However, such techniques may not always work. Nevertheless, even if an app adheres to the design guidelines, following design guidelines is not necessarily sufficient in creating high-quality apps or user experiences. In these cases, heuristics would benefit from human judgment and inference.

In addition to requiring human intervention, evaluating some heuristics may necessitate a particular level of expertise. For instance, an average smartphone user may be able to easily detect when an app has an inconsistent UI, or that an app performs poorly based on slow response. However, it may take a user with more expertise to distinguish whether the app asks for too many permissions or for unnecessary permissions. Prior research has shown that average smartphone users do not have a good understanding of the Android permission system [54, 87, 134]. For other heuristics, such as the app being malicious for deleting user data, even more expertise would be required.

A third categorization of heuristics was performed based on the level of expertise necessary to determine the output of the heuristics. This categorization was based on insights from prior work [54, 87, 134] and examinations of the range of expertise available at the interviews, which involved an information technician, 2 developers, and 9 security and usable security researchers. While the categorization has to be further validated with a more comprehensive set of interviews with participants with a range of

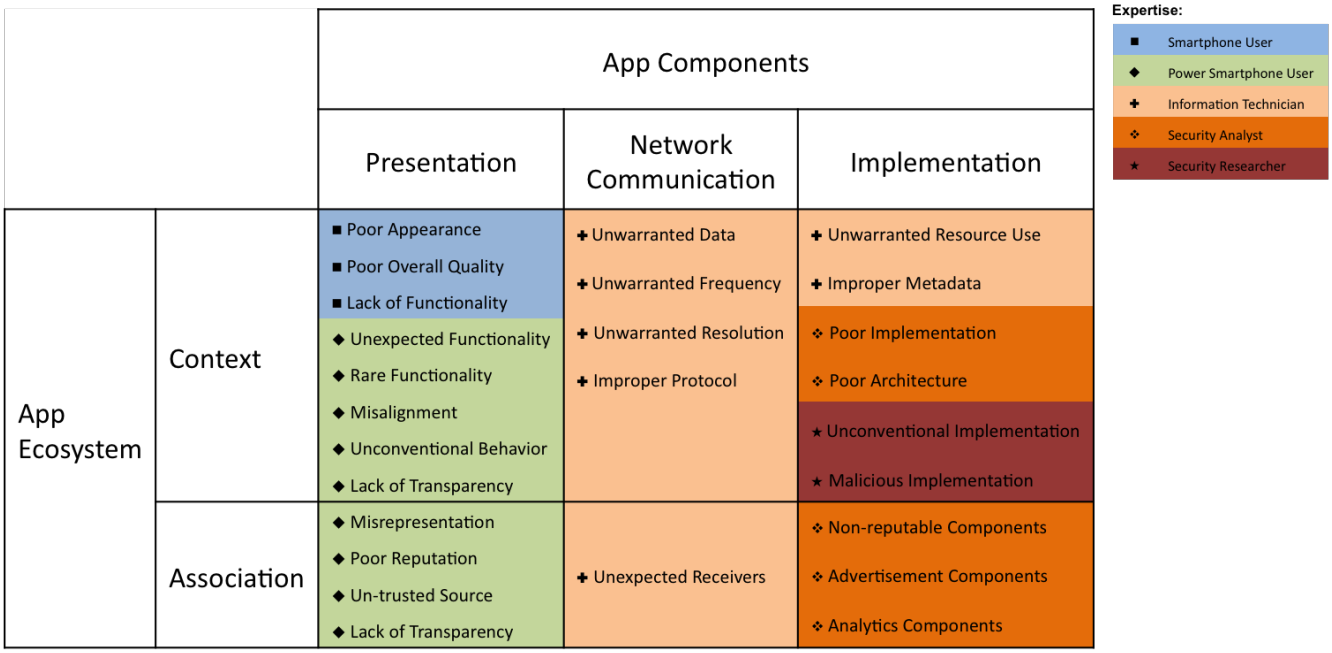


Figure 20: This figure presents a model of heuristics with respect to app components, the app ecosystem, and the level of human expertise required to determine the output of heuristics. The model offers three different dimensions to think about heuristics for finding potential app privacy problems.

backgrounds and expertise, the categorization is provided as yet another way of thinking about heuristics, especially those requiring human intervention. The expertise levels selected for this categorization are based on personas:

Smartphone User: A smartphone user who may not necessarily read or may have difficulty understanding app permission information before installing and using an app.

Smartphone Power User: A smartphone user who reads application description details and permissions and can make decisions regarding installing an app based on app permissions and her own privacy preferences.

Information Technician: Someone with knowledge of computer networking and security who could perform in an IT role at a company.

Security Analyst: Someone who specializes in computer security, understands approaches that can be used to exploit software, can use third-party tools to exploit software, however, cannot create new tools and approaches to exploit software.

Security Researcher: Someone who is an expert in computer security, who can find software vulnerabilities and write code to exploit them, and can potentially come up with new approaches to exploit software.

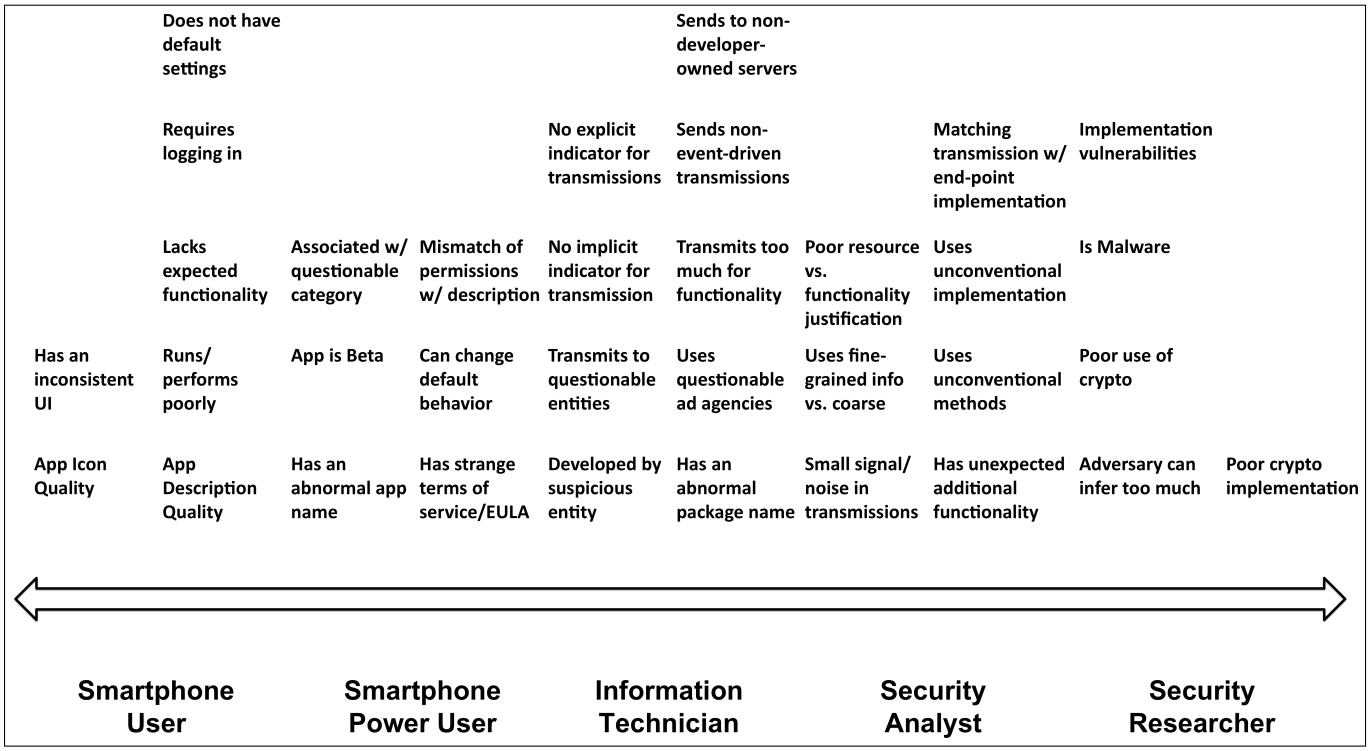


Figure 21: Not all heuristics can be performed using traditional computation techniques. Some heuristics may benefit from or even require human involvement. Determining the output of such heuristics through human inference would require different degrees of expertise. This figure presents a preliminary pass on the level of expertise necessary based on insights from prior work and the interviews. While the required level of expertise should be validated further through comprehensive studies with a diverse set of users, exploring heuristics with respect to analyst expertise is another way of thinking about app privacy heuristics. Analysts could refer to such a categorization to determine whether, for instance, crowd workers would be capable of determining the output of a heuristic or whether the analysis should be outsourced to an expert.

Figure 21 presents a set of heuristics that would benefit from human involvement vs. the range of expertise that would be required. This categorization helps privacy analysts determine what level of skill is necessary to determine the output of a particular heuristic. Analysts can use such a categorization to decide, for example, which heuristics can be performed by a crowdsourcing platform and which ones should be outsourced to experts.

5.2.4 Combining the App Components, App Ecosystem, and Expertise Dimensions

The three categorizations of heuristics based on app components, ecosystem, and expertise can be combined to present a holistic heuristics model. Such a model can be used to come up with new heuristics based on what aspect of an app is being investigated, how it relates to other apps and entities, or what level of expertise is necessary. For instance, a researcher who may be interested in crowdsourcing the output of a context-based heuristic, could use the model to see if the crowdsourcing platform available to her would

support the level of expertise needed. Another researcher may try to come up with new heuristics based on a combination of the app components involved, ecosystem aspects, and necessary expertise. The model offers three distinct dimensions to use when thinking about app privacy heuristics.

Figure 20 presents the combined model with respect to app components, ecosystem, and expertise. The *Lack of Transparency* subcategory is placed in both the context and association categories, as it applies to both. An app can lack transparency in presentation by not having a *Settings* page or options to change the default behavior of the app. With respect to Association, an app can lack transparency by not informing end-users about third-parties that may receive users' sensitive data or how users' data is used in general. The model presented is the first one to consider heuristics to find potential app privacy problems. As with all models, there may be other ways to think about the problem and categorize the heuristics.

The heuristics would be useful in identifying application privacy problems and giving more information about app behavior. A heuristics framework was built into the second version of Gort to study how people would use heuristics to find potential privacy problems.

5.3 Heuristics Framework

To enable privacy analysts to find potential problems more easily, I extended Gort to use a heuristic framework. The heuristics present privacy analysts with potential app privacy problems without requiring them to delve into the detailed operation of apps. As a result, analysts can obtain high-level information about an app's behavior without spending too much time and effort.

5.3.1 Design of the Heuristics Framework

The heuristics framework supports two different types of heuristics, namely, *static* and *dynamic* heuristics. Static heuristics are the set of heuristics that can be evaluated without having to run or interact with the app. Some example static heuristics are:

1. The app has permissions to both access the Internet and read the user's location.
2. The app can dynamically load code at runtime.
3. The app uses third-party ad libraries, e.g., *AdMob*.

Dynamic heuristics are the set of heuristics which require Gort to run and traverse the app. Through interacting with the app, Gort can learn about potential app privacy problems that occur at runtime and summarize them using heuristics.

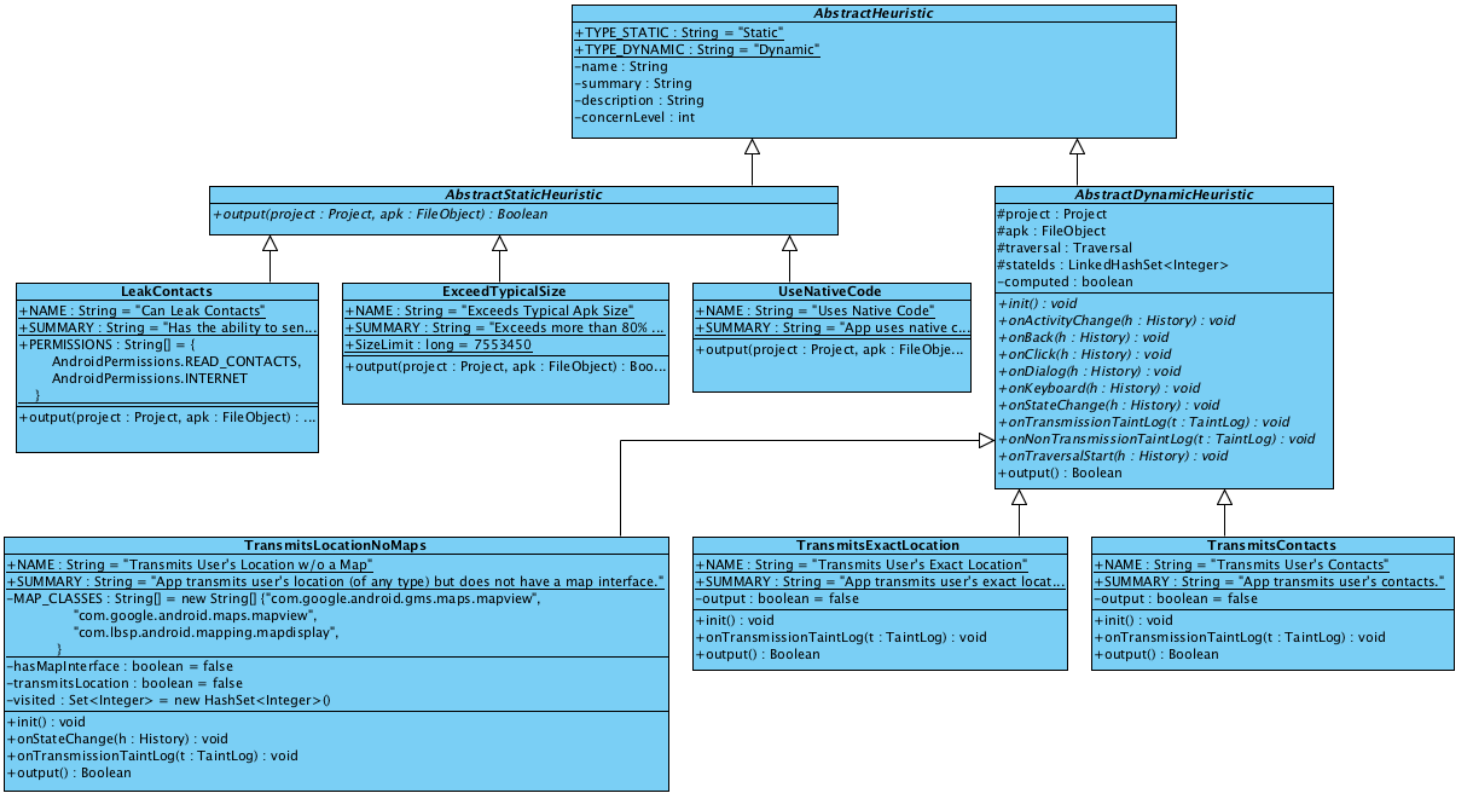


Figure 22: Privacy analysts can define and produce their own heuristics using the extendable and modular heuristics framework provided by Gort. This figure presents an overview of the *AbstractHeuristic*, *AbstractStaticHeuristic*, and *AbstractDynamicHeuristic* classes provided by Gort. These abstract classes can be extended by other analysts to create new heuristics and register them with Gort for evaluation to find potential app privacy problems. The figure also shows examples for each of the static and dynamic type heuristics.

1. The app transmits the user's location.
2. The app transmits sensitive data without encryption.
3. The app asks the user to rate it on the app market.

The heuristics framework was populated with 35 of the heuristics compiled by the author. For a list of the built-in heuristics, refer to Table 5 at the end of this chapter. In addition to the built-in heuristics, the Gort heuristics framework allows privacy analysts to define their own heuristics. Analysts can create heuristics in Java by extending the *AbstractStaticHeuristic* and *AbstractDynamicHeuristic* classes. For an overview of the classes and sample heuristics, refer to Figure 22. Gort also offers a set of APIs (e.g., hasPermission, hasAllPermissions, hasAnyPermission) and access to the traversals, metadata, and the analysis database to support creating heuristics. A design decision was made to run heuristics on recorded app traversals rather than during the traversals. First, running heuristics during a traversal would delay the duration of the traversal. Second, running the heuristics during the traversal would destabilize the traversal, especially since Gort traverses apps on a TaintDroid modified version of Android.

To create a new heuristic, an analyst has to provide a name and a description for the heuristic along with the code for Gort to execute to determine the *output* of the heuristic. An analyst would override the output function for static and dynamic heuristics and potentially override any of the event functions for dynamic heuristics, namely, *onActivityChange*, *onBack*, *onClick*, *onDialog*, *onKeyboard*, *onStateChange*, *onTransmissionTaintLog*, *onNonTransmissionTaintLog*, and *onTraversalStart*. The value of the *output* function is True, False, or Unknown depending on whether the app behaves as described by the heuristic.

5.3.2 Visualizing the Output of Heuristics

The heuristics' outputs are presented in a table as a separate information panel in the second version of the Gort GUI tool. Refer to §7.2.2 for the full details on the GUI tool's design. An analyst can sort the heuristics by their summaries or their output values. Longer summaries are truncated, although, the analyst can obtain the full summary of a heuristic by positioning the mouse pointer on the desired heuristic. By double clicking on a heuristic, the analyst can obtain the fully detailed description of the heuristic. Refer to Figure 24 for an example view of the Heuristics information panel for the Horoscope app.

Dynamic heuristics are also presented as part of the main State Flow Graph (SFG) view. Specifically, when an offending state is selected by the analyst, the analysts can view the network request, servers, permissions, and the heuristics associated with the state. When the heuristics tab is selected in the main SFG view, the heuristics that marked the app unsafe based on the particular state are presented. Figure 23 shows the heuristics for an offending state in the Horoscope app.

5.3.3 Evaluation through User Study

I evaluated the usability of heuristics as a part of the Gort v2 user study. Full details on the user study methodology and results are presented in §7.2.4. When participants were asked if they found the heuristics useful on a 5-point Likert scale (5 Strongly Agree), participants were generally positive with a median response of 4, Agree.

During the user study, it was noticed that some participants relied too much on the heuristics without looking into the application details. For instance, when asked if an app used the device camera, some participants made their decision based on the *Transmits User's Camera Data* heuristic rather than checking for the app having the *CAMERA* permission. This may result in false inference in the case where an app uses the camera but does not transmit any camera data. Similar observations were made when participants were asked if an app can send or receive text messages.

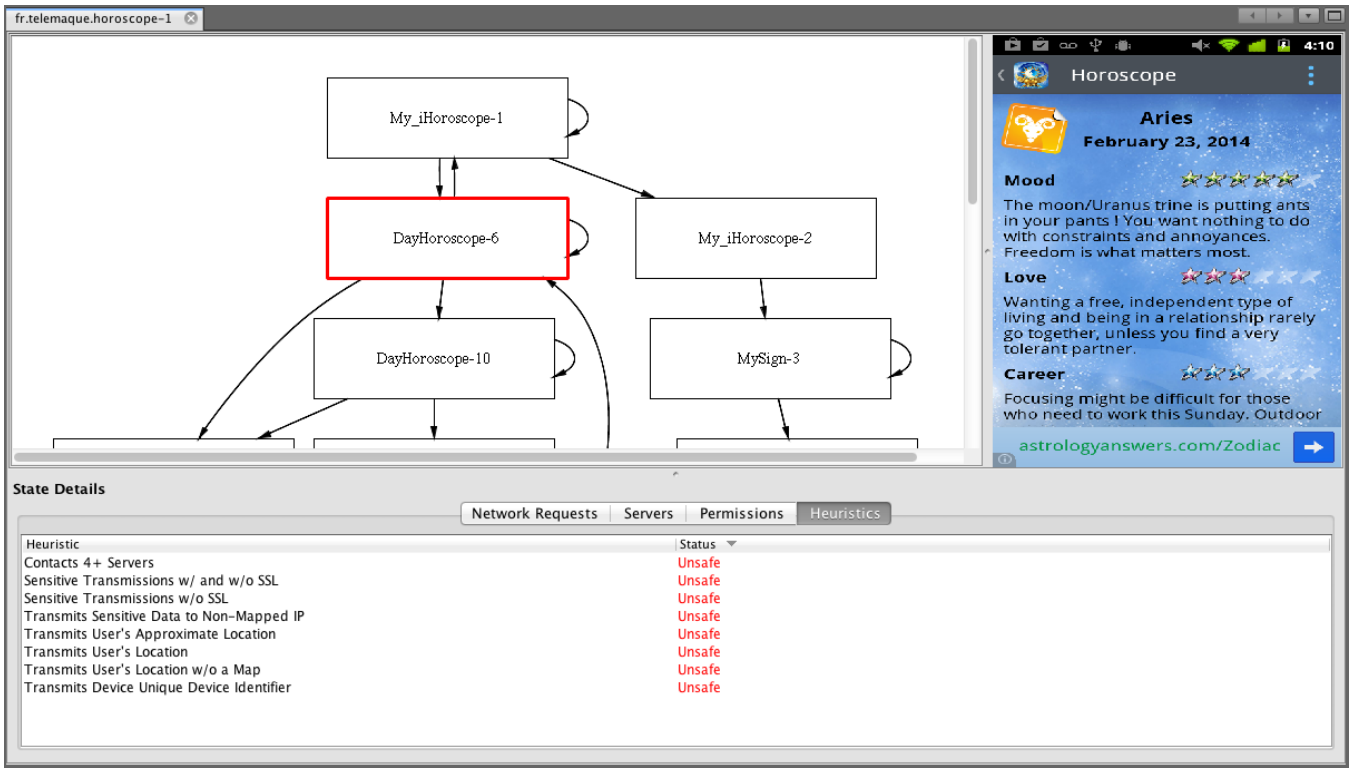


Figure 23: Dynamic heuristics that flag an app as unsafe are presented for each of the app's offending states in the State Flow Graph view. This figure shows a zoomed in view of the State Flow Graph region in Gort. The depicted view presents the heuristics for an offending state in the Horoscope app.

It was observed that participants would spend time searching through the heuristics panel for a desired heuristic. This may have been an artifact of the heuristics and their output values being presented in a table view. Several participants asked for search functionality to easily find a desired heuristic.

5.3.4 Discussion of User Study Results

It is especially important for privacy analysts to understand the meaning and output of the heuristics. As noted, some participants would rely too much on the heuristics, e.g., would assume an app does not use the device camera if a heuristic reported that the app does not transmit camera data. Prior to using heuristics, analysts should go through training to clearly understand the meaning of heuristics. It is equally important for heuristic developers to clearly define heuristics and what they mean. Moreover, it may be beneficial to also present a confidence value for each heuristic output. For instance, if a dynamic heuristic reports that the app does not transmit location, it could be that the traversal algorithm never reached the state that transmits location information. This confidence interval could be based on the app traversal coverage or other metrics.

During the study, it was observed that participants spent time searching for desired heuristics in the heuristics panel. Future designs can provide additional support through a heuristics search feature, especially in cases where viewing the heuristics' outputs through a table may not be manageable. Such designs should ensure that analysts do not jump to conclusions based on limitations imposed by search queries. Another improvement would be to group heuristics based on specific categories. For instance, all heuristics pertaining to the smartphone camera (e.g., uses the camera, takes pictures, takes videos, transmits camera data) can be grouped together to provide a more holistic view. This may reduce confusion with respect to what aspect of an app a heuristic analyzes and the significance of the heuristic's output.

Heuristic	Type
Can Create App Shortcut	Static
Can Delete App	Static
Can Delete Other Apps' Shortcuts	Static
Can Eavesdrop On Calls	Static
Can Eavesdrop On Phone State	Static
Can Install App	Static
Can Leak Coarse Location	Static
Can Leak Contacts	Static
Can Leak Precise Location	Static
Can Read System Log	Static
Can Receive And Drop SMS	Static
Can Send And Obfuscate SMS	Static
Can Set App Debug Mode	Static
Exceeds Typical APK Size	Static
Load Dynamic Code	Static
Uses Java Reflection	Static
Uses Native Code	Static
Contacts 4+ Servers	Dynamic
Sensitive Transmissions w/ and w/o SSL	Dynamic
Sensitive Transmissions w/o SSL	Dynamic
Transmits Device Unique Device Identifier	Dynamic
Transmits Sensitive Data to Non-Mapped IP	Dynamic
Transmits User's Accelerometer Data	Dynamic
Transmits User's Account Information	Dynamic
Transmits User's Approximate Location	Dynamic
Transmits User's Camera Data	Dynamic
Transmits User's Contacts	Dynamic
Transmits User's Exact Location	Dynamic
Transmits User's History	Dynamic
Transmits User's Last Location	Dynamic
Transmits User's Location	Dynamic
Transmits User's Location w/o a Map	Dynamic
Transmits User's Phone Number	Dynamic
Transmits User's Recorded Audio	Dynamic
Transmits User's SMS Data	Dynamic

Table 5: In addition to offering a heuristics framework for privacy analysts to define their own heuristics, Gort has 35 built-in heuristics to identify potential app privacy problems.

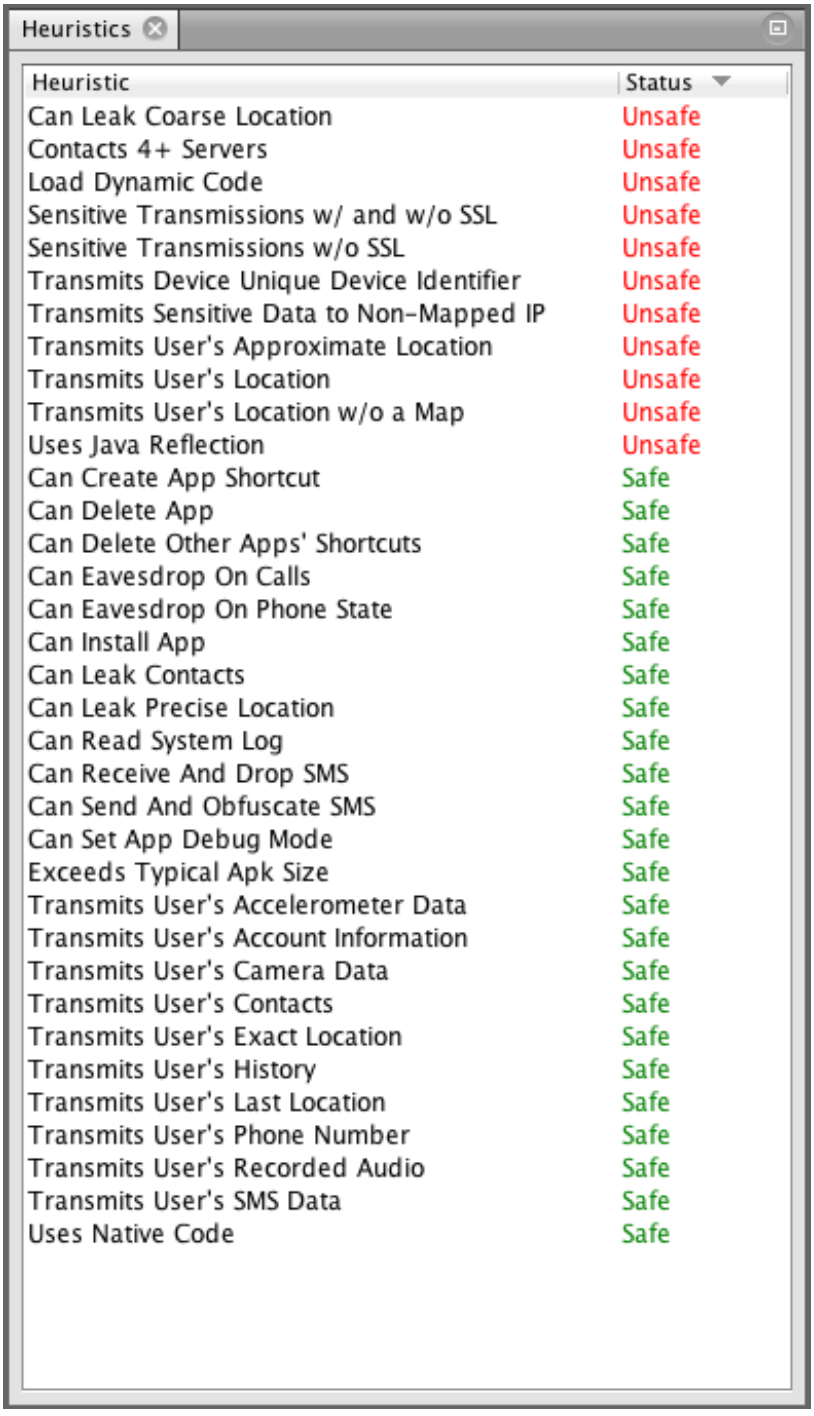


Figure 24: The output of the heuristics is presented as a separate information panel in Gort v2. The screenshot shows the output of the heuristics for the Horoscope app. An analyst can sort the heuristics by their summaries or their output values, safe and unsafe. Unsafe indicates that the app engages in the behavior described by the heuristic. Safe indicates that Gort did not find the app exhibiting the described behavior. By positioning the mouse pointer over a heuristic, the analyst can get additional summary information regarding the heuristic. Double clicking on a heuristic presents the analyst with a detailed description of the heuristic. Dynamic heuristics that flag an app as unsafe are also presented for each of the app's offending states.

6 Detecting Undesirable Privacy Leaks through Crowd Analysis

As covered in §2, a number of works already look at static and dynamic analysis of mobile apps to provide more thorough analysis for apps or provide users with protection mechanisms. While these methods are useful in detecting potentially sensitive app behavior, they cannot distinguish whether the use of sensitive resources and transmissions by an app is legitimate and beneficial to the user or intended for nefarious purposes. As an example, sending the user's location for a navigation app would make sense but may not be justified for a game app. Approaches such as TaintDroid [43] can detect when an app transmits the user's location to an external service, however, such techniques are not capable of identifying whether the transmission is necessary or beneficial to the user.

As of this writing, there are no fully automated solutions to solve this problem. Since the legitimacy of sensitive resource use and transmissions are dependent on app context and semantics, such a decision may always require human intervention. If a trained privacy analyst were to perform the evaluation, the need for additional human intervention would cost valuable analyst time. In mid 2010, I presented the first approach to solve this problem. Specifically, I proposed the use of crowdsourcing to understand whether an app's use of sensitive resources is justified based on its features and the tasks it supports. Appendix A presents the original idea of using crowdsourcing to find the legitimacy of sensitive resource use by mobile applications. Later on, this idea was fleshed out further in a technical report [9].

6.1 Early Results of the Proposed Approach

The first work which directly resulted from my proposal focused on understanding users' mental models of mobile privacy through the use of crowdsourcing [93]. Specifically, this work presented participants on Mechanical Turk with a mobile application including its name, description, and one of two study conditions, namely, *expectation* and *purpose*. The expectation condition asked participants whether they would expect the application to access a sensitive resource on their device, for instance, precise location. The expectation condition would then ask the participants to select why they think the resource was used, specifically, for functionality, advertisement or market analysis, to tag photos or other data, share with friends, or other reasons. Finally, participants were asked about how comfortable they were with the app accessing the sensitive resource. The purpose condition presented the specific reason why the associated resource was used by the app and then asked the participants whether they felt comfortable letting their device access the sensitive resource. The work had several important conclusions based on this methodology. First, users' comfort level with an app using sensitive resources is directly correlated ($r = 0.91$) with whether they would expect the resource to be used. Second, users become more comfortable with an app when they know more about how sensitive information is used by it. Third, participants on Mechanical Turk have a hard time trying to figure out why an application uses certain resources.

The first two conclusions make sense intuitively. Specifically, when a user's mental model (expectations) is closer to the world model, the user is more comfortable. In other words, people do not like being surprised when it comes to mobile app privacy. Moreover, as the user's mental model is expanded through the introduction of information about the real world (or why apps would ask for a sensitive piece of information), the user's comfort increases. Nevertheless, the third conclusion, that participants on Mechanical Turk have a hard time deciding why an application uses a certain resource, is of particular interest to this thesis. The work presented here offers a different approach which uses crowdsourcing to implicitly derive reasonable answers regarding resource usage and legitimacy of use, resulting in two important implications. First, even though the crowd may not be able to directly decide on appropriate resource use, crowd responses can be used implicitly to distinguish whether sensitive resource use is legitimate or not. Second, this work shows how a non-technical crowd can provide answers to highly technical questions.

6.2 Overview of Gort's Crowd Analysis Approach

Gort's approach to determine the legitimacy of resource use by apps through crowdsourcing works in three phases. First, Gort extracts the tasks that can be conducted using an app. In this phase, Gort presents crowd workers with screenshots of the app. It then asks the crowd workers to provide a label for the task that could be performed using the screens, in the order that they are presented. This phase is called *task extraction.* In the second phase, *task verification*, Gort presents crowd workers with the labels produced in the first phase and asks crowd workers to rate how well the label defines the task that could be performed using the screens on a 7-point Likert scale. The results from this phase allow Gort to select the best label for the task using a voting algorithm. Finally, in the third phase, Gort asks the crowd whether it makes sense for the app to use the resources that were detected by dynamic analysis while performing the task associated with the voted label. The final phase is called *resource justification*. Figure 25 shows an overview of Gort's Crowd Analysis approach.

The *extraction-verification-justification* pattern presented here enables app analysis systems to not only detect the use of sensitive resources but also determine whether the use of a sensitive resource is legitimate or not. The obtained crowd results can be used to study individual apps by privacy analysts and developers or by a large app analysis engine to flag suspicious apps. The approach does not only rely on breaking down tasks for crowdsourcing but also presents questions to crowd workers at a higher level of abstraction. Using a higher level of abstraction hides the unnecessary low level details usually offered by analysis tools. Note that while this work applies the extraction-verification-justification pattern to evaluate mobile application privacy, this pattern could be used for a variety of other problems, for instance, UI design for mobile or desktop computing environments.

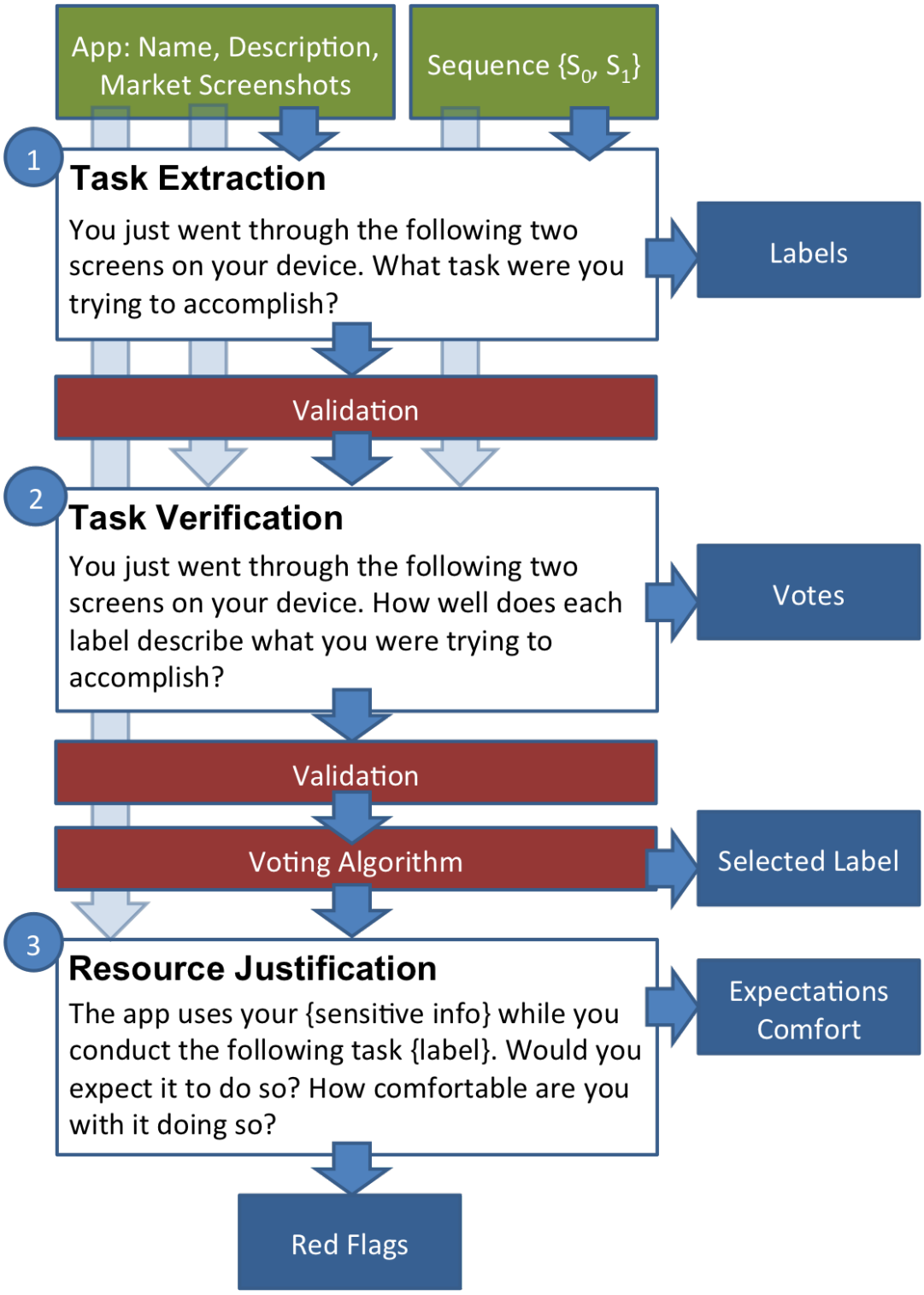


Figure 25: Gort determines the legitimacy of app privacy leaks using the extraction-verification-justification crowdsourcing pattern. During the task extraction phase, Gort asks crowd workers for labels for a pair of the app screenshots sequence. During task verification, Gort obtains votes for the labels obtained and selects the best label using the votes. Finally, in the resource justification stage, Gort asks crowd workers about their expectations and comforts with the app using the set of sensitive resources associated with the task. Based on the expectation and comfort data, Gort produces red flags for suspicious privacy leaks.

6.3 Extracting Tasks Supported by Mobile Applications

As prior work has shown, crowd workers are not able to explicitly decide on the purpose for which an app uses a specific resource. As such, it would be difficult to directly ask crowd workers whether it is legitimate for an app to use certain sensitive resources, e.g., location, contacts information. Based on my results using the extraction-verification-justification pattern, one approach that works well is to ask crowd workers whether they would expect an app to use certain resources while the user performs a specific task with the app. For instance, would they expect the app to use the user's location while the user is searching for nearby restaurants? Or would they expect the app to use the user's location while the user is reading horoscope information?

Nevertheless, to ask questions regarding application tasks and resources, it is necessary to determine what tasks can be conducted with the application and what resources the application uses while performing these tasks. Gort already records application states and monitors which sensitive resources are being used at each state through TaintDroid. As such, Gort already provides part of the input to the crowd. The more difficult input to obtain is the tasks that can be performed with the app. Android applications do not explicitly declare a list of tasks that can be conducted with the application, as such, it is important to somehow derive what tasks the app supports.

There are a number of ways to extracts tasks based on the application and application metadata. Here, I outline two approaches:

1. Android developers provide a description of their application on the Google Play store. This description usually contains features of the application and information about tasks that can be performed with an application. One approach would be to use Natural Language Processing (NLP) techniques such as Never-Ending Language Learning (NELL) [28] and tools such as openNLP[19] to discover the meaning and semantics of the app description and metadata to extract tasks. A first stab using this method could mine for verb and noun pairs, e.g., search for restaurants, take notes, tag a song. WHYPER by Pandita *et al.* is a similar solution in this space, where the authors try to find permission sentences, i.e., sentences that justify the use of resources, in an app's description available on the Google Play store [108].

2. Examining a sequence of screenshots from an application may provide enough details regarding a task or tasks that can be performed using a mobile application. Developers and designers follow certain design patterns when creating websites and apps. For instance, the 3-click rule is an unofficial best practice in web design, suggesting that a user should be able to find any information on the site with no more than three mouse clicks [135]. It is likely that, in a similar fashion, app designers and

[19] `https://opennlp.apache.org`

Figure 26: This figure presents two screenshots from the Horoscope app. The left screenshot is the main screen of the Horoscope app. The right screenshot is a screen reached from the main landing screen by a single tap interaction from the user. Even though the sequence of the screens may not explicitly inform what task can be accomplished using the two screens, the context and semantics of the screens implicitly provide that while visiting these two screens, the user would be checking her Horoscope.

developers attempt to minimize the sequence of interactions that a user has to perform to conduct a certain task with an app. As such, app context and semantics would carry information regarding the tasks potentially supported by the application. Examining applications screens to extract tasks is a potential approach, albeit human inference may be required. Figure 26 illustrates a sequence of application screens that implicitly describe that a user could check her horoscope using the two depicted screens.

A more elaborate setup may take advantage of a combination of the above and additional methods to extract the tasks supported by an application. For instance, one could attempt using computer vision techniques, such as Optical Character Recognition (OCR), on sequences of app screens to read text content and further infer the tasks using NLP techniques. Yet another method would be to search the app's APK for strings or phrases matching verb and noun pairs.

6.3.1 Crowdsourcing App Screens to Extract Tasks

Gort extracts application tasks using the second approach described above. Namely, Gort attempts to infer tasks supported by the app by presenting app screenshot sequences to crowd workers and asking for labels for the task supported by the screens. For instance, Gort would present crowd workers with screenshots similar to the ones from Figure 26 and ask for a task label.

Following the completion of the dynamic analysis phase, Gort creates a Mechanical Turk task for each app state pair that has sensitive network transmissions. The decision to only crowdsource state pairs with sensitive application transmissions is based on the following two criteria:

1. Long sequences of screenshots would result in more combinations of screens, which would not be a cost effective approach for crowdsourcing. On the other hand, showing a single screen may not be enough to convey the context and semantics of the app at the time of transmission. As such, showing a pair of screens is a reasonable compromise. In a sense, employing only a short sequence of interactions rather than the entire interaction sequence implies a Markov property assumption [74]. Showing two screenshot sequences is similar to a first-order Markov chain, where transition to any future state (e.g., app screen) depends only on the state at current time rather than any state from time zero (e.g., when the app was launched) up through the current time.
2. Only state pairs with sensitive transmissions are used. Unless one is interested in task extraction as an end in itself, rather than for evaluating certain aspects of an application (e.g., privacy, design), it is more cost effective to only crowdsource the tasks necessary for the evaluation.

For each state pair, Gort creates a Mechanical Turk HIT where it presents crowd workers with the application name, description, screenshots from Google Play, and asks the crowd to input their Android device version, whether they have used the app, the type of the app, and a label for a task sequence depicted by the state pair screenshots. Table 6 shows the labels from an actual HIT conducted on Mechanical Turk for the screen sequence shown in Figure 26. Appendix C presents the HIT to which crowd workers responded.

6.3.2 Validating Crowd Responses

Once the HIT published to Mechanical Turk reaches the maximum number of requested responses, also known as assignments in Mechanical Turk jargon, responses are downloaded for analysis. It is not unusual for crowd workers to attempt to game crowdsourcing platforms to maximize their benefit while minimizing workload. This is especially true on platforms with monetary compensation, such as Mechanical Turk [39, 89]. To address this problem, Gort filters responses using a simple validation algorithm to select only

Android Version	Used App	App Category	Task Label
4.2	No	Non-Game	check my horoscope
4.3	No	Non-Game	checking out your life in the future
2.3.3	No	Non-Game	Find out astrological predictions for the Aries sign for today.
4.3	No	Non-Game	See what Aries says
4.4.2	No	Non-Game	selecting your sign
4.1.2	No	Game	Get more info about aries.
4.1.2	No	Non-Game	I can click on my specific horoscope
4.1.2	No	Non-Game	Read my horoscope
Jelly Bean 4.3	No	Non-Game	checking my horoscope
4.2.2	No	Non-Game	Daily reading

Table 6: This table shows the crowd responses to a task extraction HIT for the screen sequence from the Horoscope app, shown in Figure 26. Crowd workers are asked for the version of Android that they run on their own Android device, whether they have used the app before, the category that the app would fit in, and a task label for the screen sequence. The 6th and 9th answer would be removed by the validation algorithm as the 6th response incorrectly marks the application as a Game application and the 9th answer also includes the version name rather than just the number.

the best answers for the creation of Task Verification and Resource Justification HITs. Specifically, Gort checks the following:

1. All required HIT questions must have responses.
2. The Android Version for the crowd worker's device provided by the crowd worker must be a valid version number[20]. The HIT asks workers to only provide a numerical input separated by dots and also informs workers about how to obtain this information on their device.
3. The application type must be correctly selected from *Game*, *Non-Game*, and *Book, music, or video* types.
4. The task label provided must be at least two words long.

The responses that match all the criteria above are used to create future HITs. Note that the validation steps 1-3 are also used for Task Verification and Resource Justification HITs.

6.4 Verifying Task Extraction Results

The quality of results obtained thorough crowdsourcing may exhibit high disparity [39, 89]. One approach that researchers have exercised in the past is to use crowdsourcing iteratively to check previously obtained results [21, 94]. In the Task Verification step, Gort adopts a similar approach to verify the crowd responses

[20] `https://source.android.com/source/build-numbers.html`

4. Select how well each of the options below describes what you were trying to accomplish **over the course of the above two screens**. Please take into account grammar and spelling. Note: If the two screens are too similar, please respond with respect to a single screen. (required)

	Extremely Poor	Below Average	Slightly Below Average	Average	Slightly Above Average	Above Average	Excellent
check my horoscope	○	○	○	○	○	○	○
checking out your life in the future	○	○	○	○	○	○	○
Find out astrological predictions for the Aries sign for today.	○	○	○	○	○	○	○
See what Aries says	○	○	○	○	○	○	○
selecting your sign	○	○	○	○	○	○	○
Get more info about aries.	○	○	○	○	○	○	○
I can click on my specific horoscope	○	○	○	○	○	○	○
Read my horoscope	○	○	○	○	○	○	○
Daily reading	○	○	○	○	○	○	○
Select "Above Average" here	○	○	○	○	○	○	○

Figure 27: This figure shows the target question for the Task Verification HIT produced for the screen sequence depicted in Figure 26. Note that the 'Get more info about aries' label is still included as an option as the actual implementation does not validate app types and only relies on the remaining validation methods.

from Task Extraction to select the best label for a given app screen sequence. Namely, Gort requests a second set of crowd workers to vote on the quality of responses from the Task Extraction step.

After filtering poor quality answers using the validation method, Gort combines all the labels from Task Extraction and creates a Task Verification HIT. Similar to the Task Extraction HITs, Task Verification HITs contain questions asking participants for the Android device version number, whether they have used the app, and the app type. These questions are used to validate the HIT results. However, the target question for Task Verification is a table that presents each of the label options and asks crowd workers to rate their quality on a 7-point Likert scale. Figure 27 shows a sample Task Verification target question. Note that the 'Get more info about aries.' label is still included as an option as the actual Gort implementation does not validate the app category type question and only relies on the remaining validation questions.

The Task Verification HIT requests 10 crowd worker assignments from Mechanical Turk. Following this step, Gort verifies the crowd responses again using the same validation approach outlined in §6.3.2. At this point, a voting algorithm may be used to select the 'best' label for the associated application screen sequence. For this purpose, Gort uses a *Majority Judgment* voting algorithm [17]. This algorithm was chosen as its use of medians makes it robust to outliers. Specifically, voters freely grade each one of the candidate labels for the task on a 7-point Likert scale. The candidate with the highest median grade is selected as the best label. In the case of more than one candidate with the same median, the algorithm uses a tiebreaker to see the 'closest-to-median' grade. However, Gort simply chooses the first option with highest median in such cases. Running the Majority Judgment voting algorithm on the results from the actual Mechanical Turk on the aforementioned screen sequence resulted in 'check my horoscope' to be selected for the task label.

6.5 Determining the Legitimacy of Sensitive Resource Use

The result of the Justification stage is the end goal of the crowd analysis. The Justification step combines the results from the dynamic analysis of the app and the Task Extraction and Verification stages to produce a new HIT. Note that while the results from the Task Extraction and Verification stages are treated as intermediary for the purposes of Gort, there may be other use cases where acquiring labels for a task or finding a best label for a task may be an end in itself. Moreover, others may use the extraction-verification-justification pattern for different purposes, e.g., usability testing. Gort uses the Justification step to determine the legitimacy of sensitive resource use by apps.

The Resource Justification HIT asks two target questions in addition to the validation questions of Android device version, prior app use, and application category. The two target questions ask crowd workers whether they would expect the app to use the resources that it does while conducting the task selected by the Task Verification stage. Furthermore, the HIT asks crowd workers about how comfortable they feel with the app using the resources for the task. Figure 28 shows the target questions asked for the screen sequence depicted in Figure 26.

Similar to the other stages, the Resource Justification stage asks Mechanical Turk for 10 assignments for the HIT. Once results are obtained, a number of measures can be computed for the crowd expectations and comforts. At this point, Gort calculates a median based on the expectations to see whether crowd workers would expect the app to use the sensitive resources that it does. Gort also calculates a mean, median, and standard deviation for the comfort levels for the task and resources combination. While these measures are only used in the Gort tool to be presented to privacy analysts, they could be used by an automated Analysis Engine or by machine learning techniques to auto grade app privacy or to recommend privacy-preserving apps to end-users. For the 'check my horoscope' task above with the 'phone number,

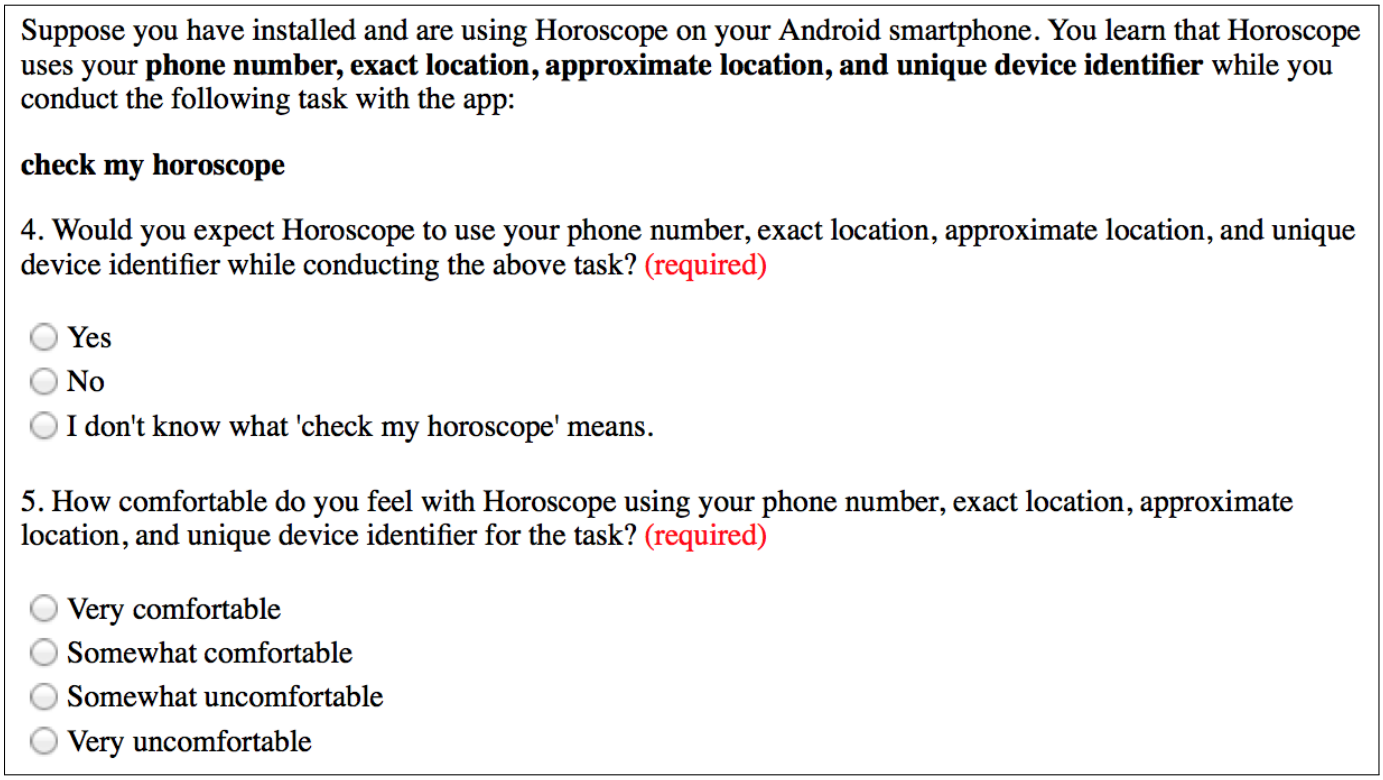
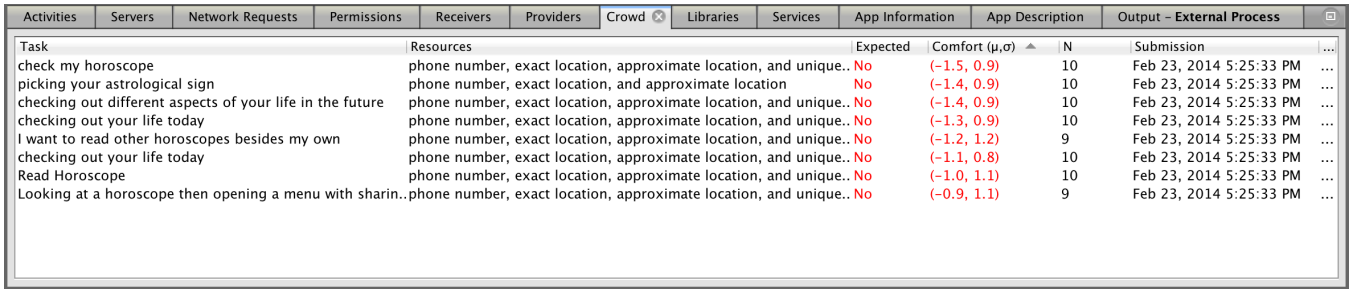


Figure 28: This figure shows the target question for the Resource Justification HIT produced for the screen sequence depicted in Figure 26. The HIT asks crowd workers about their expectations and comforts regarding the app using the sensitive resources mentioned while performing the task selected during the Task Verification stage.

exact location, approximate location, and unique device identifier' sensitive resources in use, the median expectation was 'No' with μ=-1.5 and σ=0.9.

6.6 Crowd Analysis Implementation

Crowd analysis is implemented as part of Gort. After an app has been traversed, the privacy analyst may choose to perform crowd analysis on the app. During crowd analysis, Gort reviews the traversal for state transitions. For each transition, Gort checks to see if either state has had sensitive transmissions. If neither state has sensitive transmissions, then Gort moves to other entries. For transitions that have sensitive transmission in at least one state, all sensitive information types are recorded and combined for both states. This list of resources is what is presented to the crowd during resource justification. The design rationale for combining sensitive resource types is that if each state pair and resource combination were to be crowdsourced, crowdsourcing analysis for an app would be too costly and not scalable. For sensitive transmissions that occur in one state, Gort only presents the crowd with a single image for the HIT, and updates the HIT question to focus on a single screenshot.



Task	Resources	Expected	Comfort (μ,σ)	N	Submission	...
check my horoscope	phone number, exact location, approximate location, and unique..	No	(-1.5, 0.9)	10	Feb 23, 2014 5:25:33 PM	...
picking your astrological sign	phone number, exact location, and approximate location	No	(-1.4, 0.9)	10	Feb 23, 2014 5:25:33 PM	...
checking out different aspects of your life in the future	phone number, exact location, approximate location, and unique..	No	(-1.4, 0.9)	10	Feb 23, 2014 5:25:33 PM	...
checking out your life today	phone number, exact location, approximate location, and unique..	No	(-1.3, 0.9)	10	Feb 23, 2014 5:25:33 PM	...
I want to read other horoscopes besides my own	phone number, exact location, approximate location, and unique..	No	(-1.2, 1.2)	9	Feb 23, 2014 5:25:33 PM	...
checking out your life today	phone number, exact location, approximate location, and unique..	No	(-1.1, 0.8)	10	Feb 23, 2014 5:25:33 PM	...
Read Horoscope	phone number, exact location, approximate location, and unique..	No	(-1.0, 1.1)	10	Feb 23, 2014 5:25:33 PM	...
Looking at a horoscope then opening a menu with sharin..	phone number, exact location, approximate location, and unique..	No	(-0.9, 1.1)	9	Feb 23, 2014 5:25:33 PM	...

Figure 29: This figure presents the Crowd tab on the Gort GUI tool for the Horoscope app. Gort monitors Mechanical Turk for result updates for each HIT. Upon completion of the Crowd analysis, Gort computes statistics such as the mean, median, and standard deviation for the crowd workers' comfort and the median for the crowd expectations for each extraction-verification-justification instance. The comfort values are in the [-2, 2] range. A comfort level closer to -2 indicates that the crowd worker did not feel comfortable with the app using the specified resources for the task.

Gort performs the steps for crowd analysis as defined above by creating HITs on Mechanical Turk. The app transition screenshots are uploaded to a Google App Engine [65] instance, which serves the screenshots to Mechanical Turk. Gort checks for results for HITs every 3 minutes. Once a task has obtained the required number of results (10 in the current implementation), the results are processed to create the corresponding next task for task verification or resource justification, according to the step for which results were obtained. When Gort obtains results for the last step, namely, resource justification, it evaluates statistics for the crowd task. The statistics consist of the number of valid answers, means, medians, and standard deviations for the crowd workers' comfort with the app's behavior, and median for crowd expectation. Figure 29 shows how these results are presented to the privacy analyst on the Gort GUI tool.

Crowd workers are offered $0.20 for answering each question regardless of question type, e.g., extraction, verification, justification. As 10 responses are required for each type, the cost of analyzing one app transition pair is $6 (3 x 10 x $0.20). In the current implementation, Gort is limited to performing crowd analysis for 16 transitions, which keeps the crowd analysis cost per app to $96, or less than $100 per app. In this thesis, some apps were analyzed for more than 16 transitions.

6.7 Evaluation through Analyzing Popular Android Apps

I evaluated crowd analysis through running the full end-to-end Gort analysis on 22 Android apps. The full end-to-end analysis refers to first performing static analysis, then traversing the apps using Squiddy while monitoring them through TaintDroid, and finally running the crowd analysis on the app.

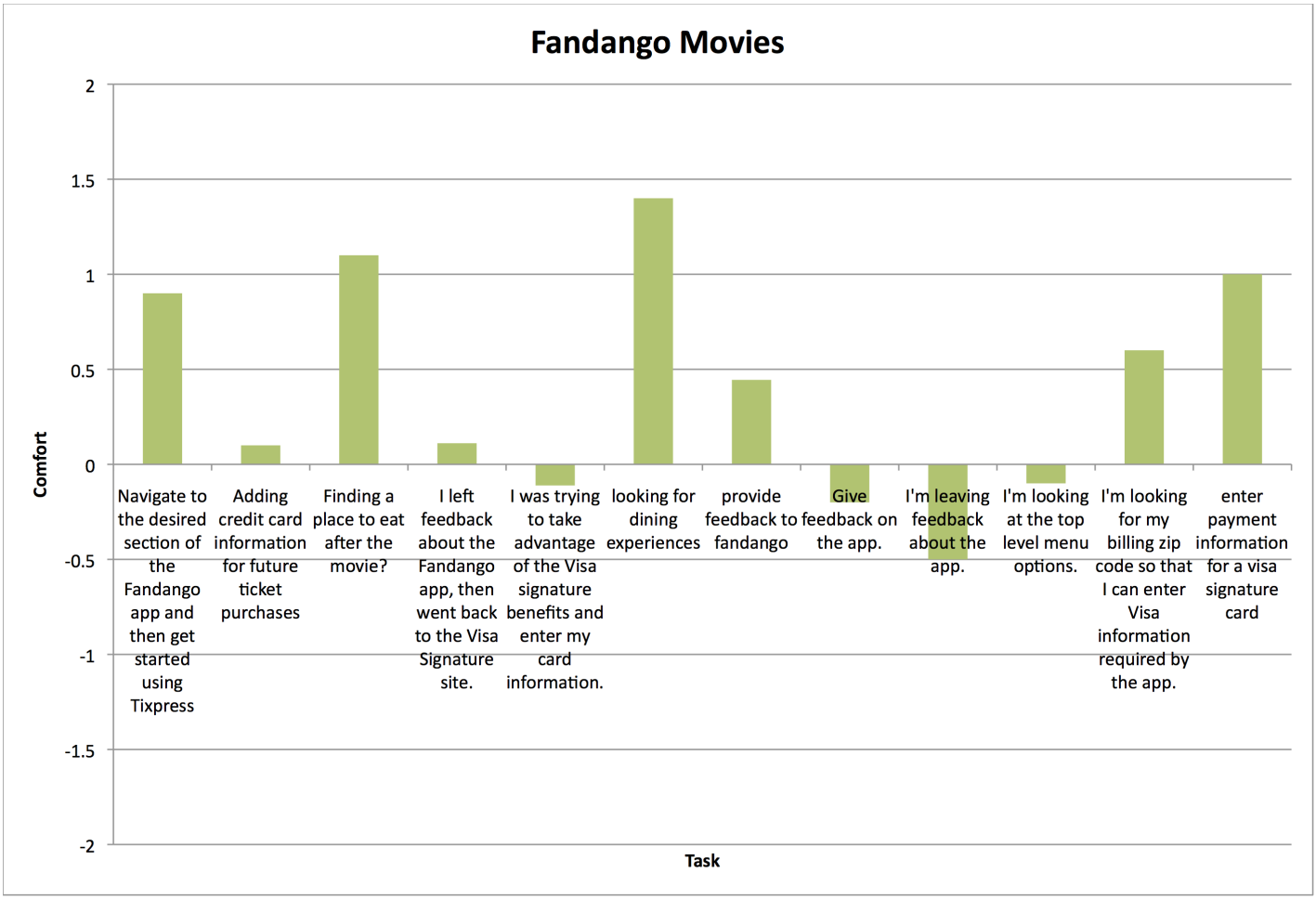


Figure 30: The Fandango Movies app sends out the users' phone number and location while conducting the tasks presented in the chart. The tasks are extracted through crowdsourcing app transitions and asking crowd workers for labels for the task. Crowd workers are generally fine with sending out their phone number and location for finding nearby information, however, are not as comfortable when the app is sending feedback or when they are entering purchase information into the app.

6.7.1 Understanding Tasks and Comfort Levels

Figures 30-32 show the tasks extracted and the comfort levels for 3 of the 22 applications, namely, Fandango, Horoscope, and Yelp. Appendix H presents comfort levels and the tasks charts for each of the 22 apps. For the reader's convenience, the charts included here are replicated in larger format in Appendix H. Crowd workers are generally more comfortable with apps using sensitive user information such as location or phone number when they are performing tasks with the app that would require such information.

In the case of the Fandango app, crowd participants are fairly comfortable with the app using their phone number and exact location when they are 'looking for dining experiences' using the app (μ=1.4, σ=0.5). However, users are not comfortable with the app using the same information, when they are

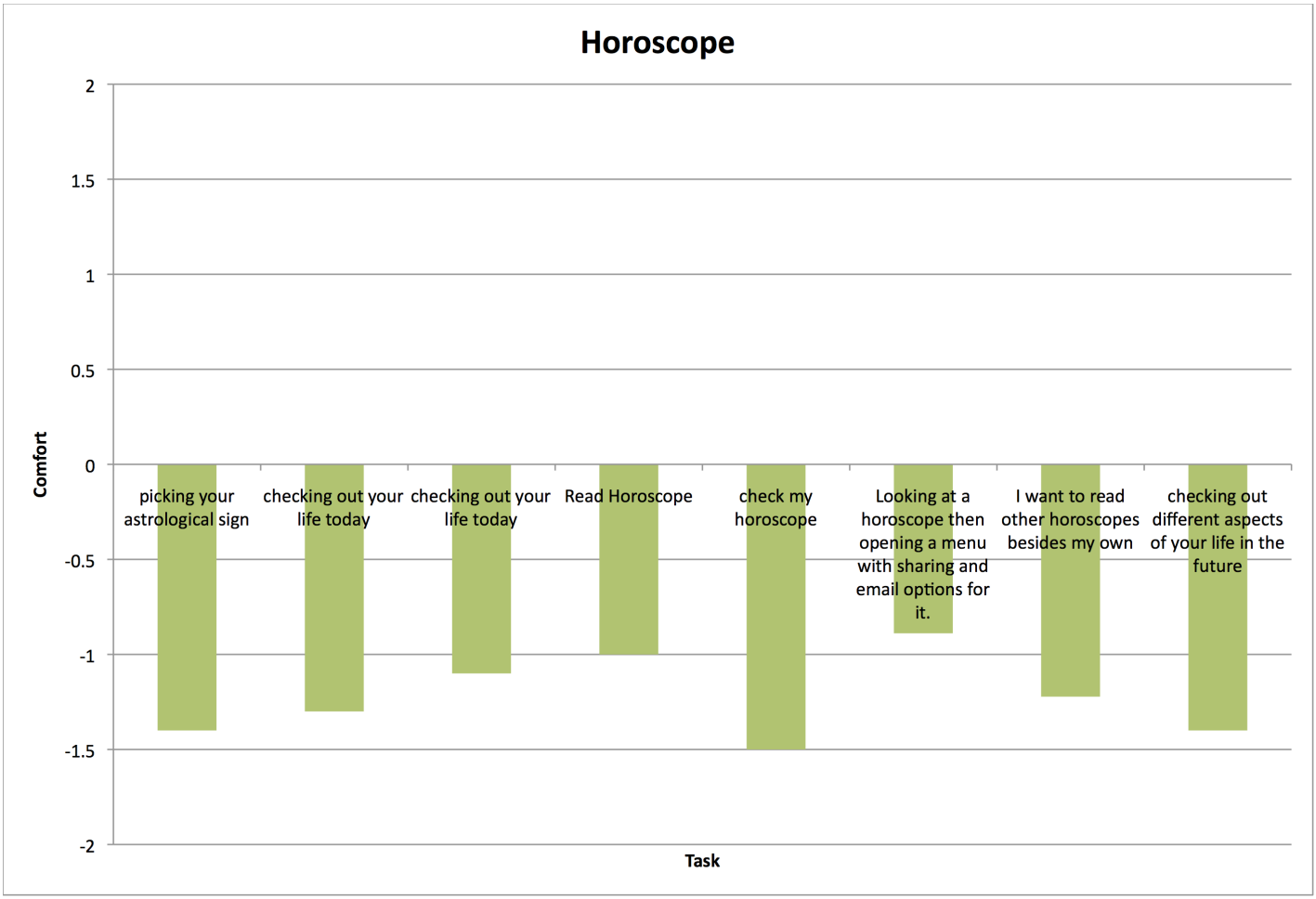


Figure 31: Horoscope sends out the users' location and unique device identifier (e.g., IMEI) while users perform the above tasks with the app. Crowd workers are not comfortable with the app using their location or device unique identifier for any of the tasks performed.

'leaving feedback about the app' (μ=-0.5, σ=1.3). The Horoscope app shares the users' location and unique device identifier (IMEI). Crowd workers are generally not comfortable with the Horoscope app's conduct. The crowd workers' comfort for the 'check my horoscope' task is (μ=-1.5, σ=0.9), which is the lowest comfort level for the Horoscope app. The highest comfort level for the app is (μ=-0.9, σ=1.1) for 'Looking at a horoscope then opening a menu with sharing and emails options for it.' For the Yelp app the lowest comfort level is (μ=-0.4, σ=1.1) to 'find directions for Newell Simon Hall' (the app shows a map centered at Newell Simon Hall where the traversal was performed). The highest comfort level was (μ=1.3, σ=0.5) to 'find restaurant best match'.

Computing the mean and the standard deviation for crowd workers' comfort level is useful not only in knowing the comfort level, but also in understanding which questions the crowd had trouble answering. When the standard deviation is high, it communicates that the crowd is wavering about answering the

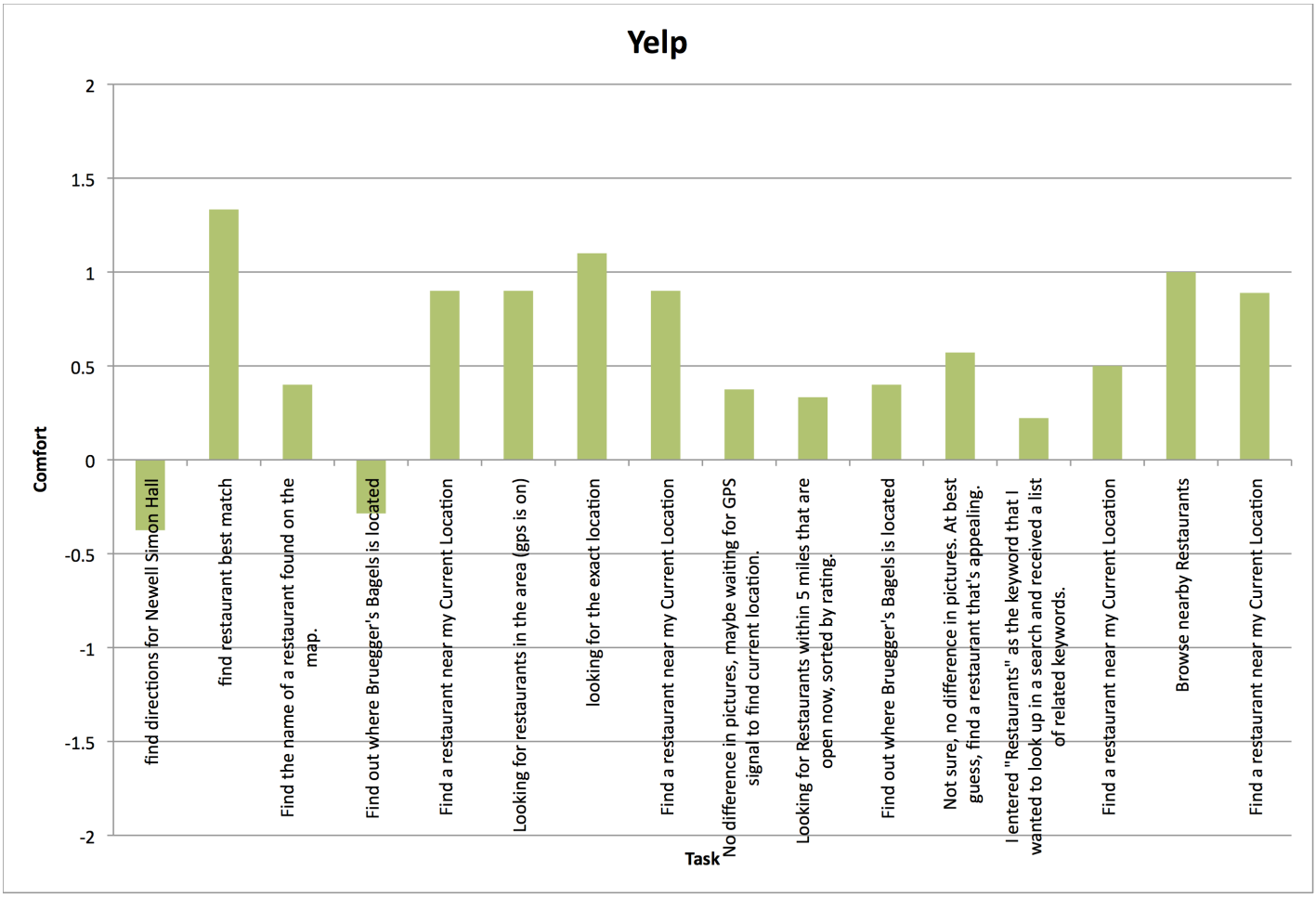


Figure 32: The Yelp app sends out a user's phone number and location while the user performs the above task with their device. Crowd workers are generally fine with the app using their phone number and location. However, for cases where the app is finding specific POIs with known locations, crowd workers are not as comfortable with having their phone number and location shared. Searching for a specific POI does not require the user's location, e.g., a search for the Eiffel Tower.

question or unsure about its comfort. Regardless of whether the difficulty is in answering the question or with the app's behavior, a high standard deviation communicates that the task and sensitive information should be inspected more closely. Here, having a human in the loop, such as a privacy analyst, would be helpful.

6.7.2 Crowd Analysis Time Performance

In addition to providing useful results for the legitimacy of app privacy leaks, it is important for crowd analysis to be feasible with respect to the amount of time and funding required for analysis. For potential funding models, refer to §1.3.1. This section provides results with respect to the duration of the analysis.

Table 7 presents the timing analysis for each of the task extraction, task verification, and resource justification steps of crowd analysis. Each of the task extraction and task verification take on average less than an hour to complete. Specifically, on average task extraction HITs take 53 minutes to complete with σ=90 minutes. Task verification HITs take 51 minutes on average with σ=106 minutes. Resource justification HITs take less time for completion, which may have to do with them being simpler. The resource justification step takes 26 minutes on average with σ=41 minutes. On average, it takes approximately 130 minutes (a little over 2 hours) for all three steps of the crowd analysis to be completed.

Comparison with the Number of Apps Added Daily: The number of apps added monthly to the Apple App Store is approximately 20,000 [116]. This figure is approximately the same for Android [11]. Based on this figure, about 667 apps are added on a daily basis. Assuming that static and dynamic analysis would take approximately 60 minutes and crowd analysis takes a full 130 minutes for an app (keeps the instance busy the entire time), a total of 89 analysis instances (cores), could cover the daily number of apps added. The 89 instance estimate is an overestimate as an instance can perform other analysis, e.g., static or dynamic analysis, while an app is being analyzed using crowdsourcing. Considering that multiple analysis engines can run in parallel, it is feasible that all apps added daily can be analyzed without overflowing to the next day. The estimate is based on the assumption that there is always a pool of available crowd workers.

App	Tasks	Extraction† (μ, σ)		Verification† (μ, σ)		Justification† (μ, σ)		Total† (μ, σ)	
Best Parking	10	12.60	1.74	16.55	10.34	12.01	6.87	41.15	12.43
Bible App	5	13.79	2.27	33.22	12.53	22.67	7.89	69.67	10.06
Calorie Counter	7	26.06	4.50	24.63	6.01	12.06	2.34	62.75	8.24
CardioTrainer	4	8.01	0.67	8.26	0.43	12.07	14.30	28.33	14.68
CBS News	9	15.69	5.97	16.81	8.10	15.20	10.15	47.71	5.98
Fandango	12	78.83	6.14	30.99	6.92	27.06	15.74	136.88	20.76
Flashlight	8	22.50	2.30	41.18	17.18	19.27	11.69	82.94	13.40
GasBuddy	14	26.51	18.14	78.83	14.99	30.82	11.62	136.16	24.81
Groupon	14	14.92	3.49	12.37	3.38	12.96	4.33	40.25	6.96
Horoscope	8	467.55	74.87	89.49	9.08	40.94	13.05	597.97	75.50
iRadar	2	21.46	0.01	30.43	8.71	19.23	13.34	71.12	4.64
KBB.com	20	31.69	4.62	14.75	5.66	8.48	1.87	54.92	5.50
MapQuest	5	12.72	1.86	12.09	3.54	5.48	2.49	30.28	2.36
PhoneInfos	14	32.64	4.55	9.69	3.30	6.51	1.41	48.84	4.98
QR Code Reader	3	36.72	2.11	20.10	10.13	20.75	4.37	77.57	9.51
Shazam	16	122.34	19.16	34.11	16.57	30.33	7.40	186.78	16.89
Speedtest.net	2	8.98	0.62	16.13	10.31	12.91	2.49	38.01	12.17
TripAdvisor	22	19.51	10.98	24.32	12.39	10.62	5.64	54.45	19.38
WeatherBug	10	145.88	55.78	521.54	83.28	200.33	38.14	867.75	114.04
WhitePages	19	13.23	2.37	17.01	2.64	8.87	1.09	39.11	3.72
Yellow Pages	2	5.95	0.78	4.19	0.53	3.33	0.75	13.48	0.55
Yelp	16	22.07	11.29	42.25	12.90	32.80	23.48	97.12	20.38
Overall‡		53.08	90.77	51.04	106.21	25.94	41.09	130.06	195.16

Table 7: This table presents the amount of time it takes for the three different stages of crowd analysis performed by Gort. ‡Time values are in minutes. †The Overall row presents the mean and standard deviation computed for the time it takes for all the tasks presented.

7 Gort Tool: An Interactive GUI for Evaluating the Privacy of Mobile Apps

As a part of this thesis, I built a new tool, Gort, to help privacy analysts evaluate mobile app privacy. Gort simplifies privacy analysis using four distinct approaches. First, it interacts with mobile apps automatically while recording app state and monitoring sensitive app behavior. This approach reduces the burden on analysts to manually install and interact with apps. Second, Gort has a set of built-in heuristics to help analysts find potential privacy problems. For example, one heuristic checks if the app has access to both the user's location and the Internet, which would allow the app to track users. I elicited over 100 heuristics by interviewing 12 experts, and have currently implemented 35 of them in Gort (§5). Third, Gort presents app behavior and heuristic results in the context of the app by presenting the states, associated screenshots, and sensitive transmissions using an easy to understand graphical user interface (GUI). The GUI enables analysts to view and digest app behavior more effectively. Fourth, through the use of crowd sourcing, Gort determines crowd comfort and expectations with specific app behaviors and also raises red flags based on the crowd's opinions regarding the app behaviors.

7.1 First Iteration of the Gort Tool

I took an iterative approach to designing Gort. I built two versions of Gort, namely, v1 and v2. Gort v1 was a preliminary version built to help understand the design space. While the first version enabled automated scanning of apps and some static analysis, it did not support the heuristics framework and the entire workflow as described in §1.4, which are supported in Gort v2. This section presents the requirements, design, implementation, and evaluation of the first version of Gort. For the purposes of this work, I focused on the Android operating system. This decision was made based on the open source nature of Android as well as the availability of tools and extensions (in particular, TaintDroid [43], and TEMA [122]).

7.1.1 Requirements for Gort v1

The requirements for designing the first version of Gort were informed by expert reviews, examination of prior related work to mobile privacy and security, study of tools for network flow analysis (e.g., WireShark), and a manual inspection of over 40 Android applications. The following are the design requirements for Gort v1.

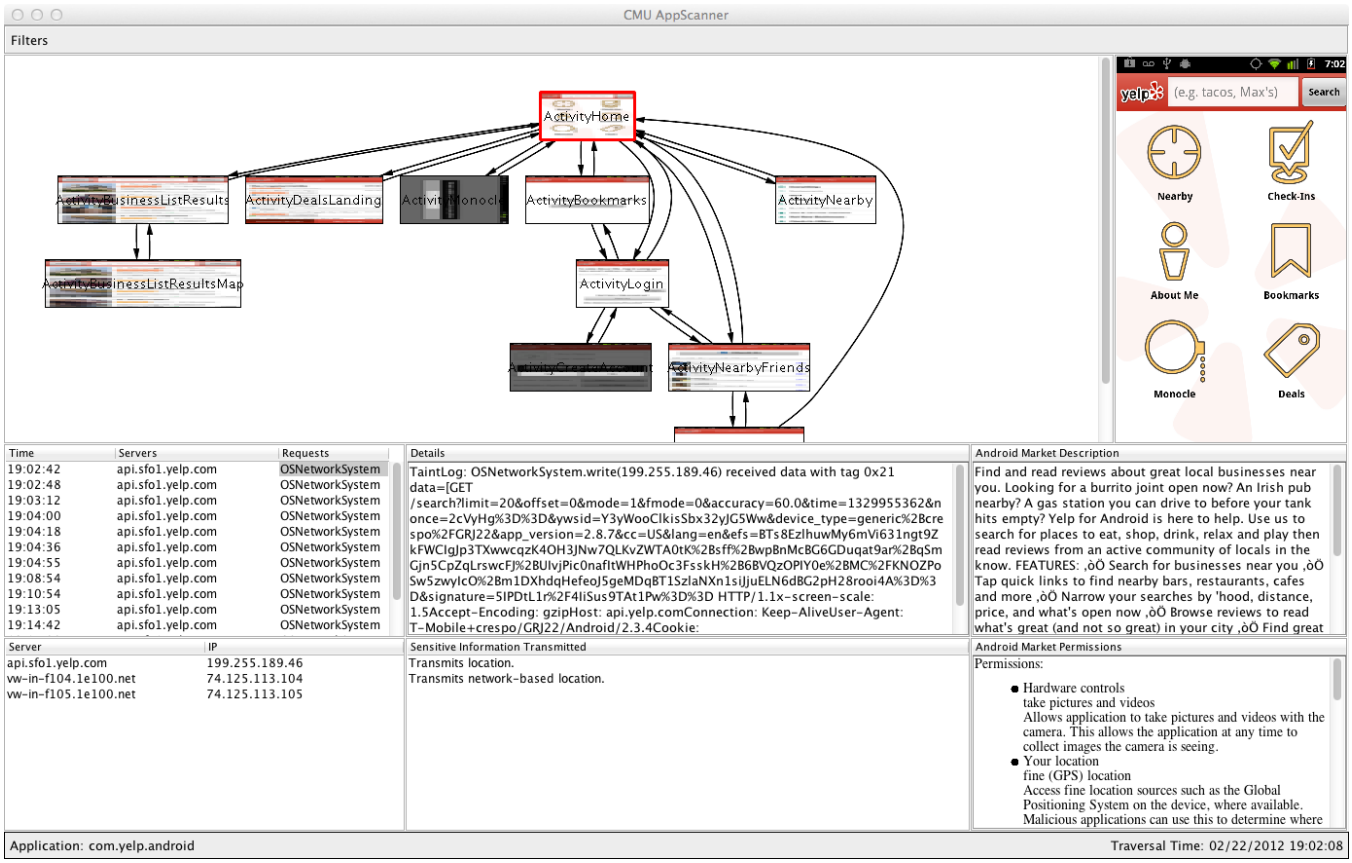


Figure 33: Gort enables application privacy analysts to learn about application behavior through an interactive GUI. The first iteration of the Gort tool, as depicted here, presents an overview of the Yelp application, which includes a high-level *Activity Flow Graph* (AFG) of the Yelp screens, all the servers contacted with sensitive information during the application run, sensitive information sent to the servers, and a list of all sensitive requests along with their details. Gort presents a screenshot of the screen whenever the user clicks on an activity shown in the graph. The GUI also presents the description of the app as available on the Google Play store along with the permissions that it declares. The analyst can define a number of filters which narrow down the visualization to interactions based on a certain type of sensitive information, communications with a particular server, or a particular request type.

Automate Application Scans: One of the barriers to scaling up mobile app inspection is having to manually interact with apps to trigger their behavior [78, 81, 126]. Traditionally, this step required analysts to interact with the live application to discover application states and inspect application behavior. When I began my study of analysis tools, there did not exist any automated application traversal solutions. Interacting with applications was either performed manually or relied on scripting languages. While a number of tools existed to support creating scripts to perform UI interactions (e.g., Robotium[21], Scirocco[22], Android Money Runner[23], and TEMA [122]), the existing solutions required expert knowl-

[21] `http://code.google.com/p/robotium/`
[22] `http://code.google.com/p/scirocco/`
[23] `http://developer.android.com/tools/help/monkeyrunner_concepts.html`

edge and imposed a learning period. As such, one of the requirements for Gort focused on automating application interactions to explore application states and trigger application behavior.

Expose Sensitive Transmissions: Traditionally, privacy analysts relied on setting up isolated networks [125] or routing application traffic through proxies [81], in order to capture sensitive application transmissions. Furthermore, analysts had to manually examine the content of network packets to identify the sensitive information transmitted. This requirement was imposed to remove the need to setup a separate network or a proxy for transmissions and to detect sensitive packets without requiring manual inspection of individual network transmissions.

Present Behavior in Context: It is important to associate sensitive application behavior with specific states of the applications. As such, Gort presents sensitive application transmissions with the corresponding state of the application at the time of transmission.

7.1.2 Design for Gort v1

Gort v1 has two major components. The first is Squiddy, which is the algorithm that interacts with mobile application UI elements to explore and discover application states while monitoring sensitive network transmissions. The second is an interactive GUI tool, which presents the results from the traversal. Squiddy is covered separately in §4. This chapter focuses on the interactive GUI tool.

It can be difficult for app developers and privacy analysts to understand the behavior of apps. The task of identifying sensitive information use, recognizing sensitive transmissions, and mapping them to application behavior is non-trivial. As a result, having access to a GUI that summarizes the information and presents it in a manageable form is highly desirable. The first version of the interactive visualization was implemented to show how the major screens of an app are connected, what resources are used and when, what remote servers the app connects with, and what data is sent to those servers. The GUI was designed to enable analysts to view the overall behavior of an app and drill down using brushing and linking, a technique that lets users select information in one view and see how information on other views is affected [15]. For example, the analyst might select a sensitive transmission type (e.g., 'Transmits location') and see what screens transmit location. The analyst might select a remote server and see all the screens where data is sent to the server plus what data is sent.

Figure 33 shows a screenshot of the first Gort GUI presenting information regarding the Yelp mobile app. The Yelp app allows users to search for nearby points of interests (e.g., restaurants), check-in to places, and find deals[24]. On the top left corner of the visualization, the user is presented with a control flow graph of the app. The AFG is interactive. Upon receiving a click on a single activity or screen, the

[24] `www.yelp.com`

GUI presents a screenshot of the activity on the top right portion. In the midsection, the GUI shows corresponding requests and request details of the selected screen. The application description and permissions, as obtained from the Google Play store, are shown on the bottom right portion. Finally, the lower left and lower mid portions of the UI show all the servers and sensitive information that are used by the application.

The GUI allows the user to manage the amount of information presented by creating filters or by linking pieces of information using brushing and linking approaches. The user can create filters based on a particular server, a specific request type (e.g., HTTP, HTTPS), or a particular resource that is involved, such that only related information corresponding to the filters are presented by the GUI. Moreover, filters can be combined in form of an intersection between categories (e.g., server and resource) or a union within categories (e.g., all the requests related to server *x* or *y*).

7.1.3 Implementation of Gort v1

The implementation of Gort v1 consists of approximately 10,000 source lines of code in Java and Python, written by the author. The implementation details for Squiddy are covered in §4. Gort v1 and v2 both use TaintDroid based on the Android platform version 2.3.4, as I found it to be the most stable version for analysis and testing. The GUI was implemented in Java using Java Swing. The interactive AFG is processed using Graphviz [71] and presented using the Grappa Java Graph Package [72]. Traversal results were stored in an isolated SQLite [77] database for each traversal.

7.1.4 Evaluation of Gort v1

I evaluated the first version of Gort using two approaches. First, I performed a preliminary analysis of Squiddy's application traversal coverage, which is covered in §4. This coverage analysis helped identify traversal hurdles and resulted in improvements in the second version of Squiddy. Second, I conducted a user study of Gort where I presented participants with a video tutorial of how the Gort GUI tool worked and then asked them to analyze a mobile application with the tool. I asked the participants about their general opinion of the tool, the features that they would add or remove, and changes that they would make to the application's layout. These results informed our requirements and design of the second version of the GUI tool.

User Study: I conducted a user study with Gort v1 at the same time as the heuristics interviews with the same set of participants. For the reader's convenience, the breakdown of participants is repeated here. The user study was performed with 12 participants, consisting of 9 security and usable security researchers, 2 developers, and 1 IT professional. 3 of the participants were female and the other 9 were male. 10 out of the 12 participants were between 20-29 years old. Recruitment focused specifically on people who had a

background in security, development, and IT and prior experience using smartphones. Participants were compensated with $30 Amazon gift cards.

During the study, participants were presented with the Gort GUI and shown how to use the tool. After this step, each participant was asked to evaluate an application (Yelp, Fox News, Pulse News, Horoscope, TossIt, ColorFind) using the Gort tool and provide information about what types of sensitive transmissions were sent out, when transmissions were made and to whom. Participants were also asked to identify potentially privacy-intrusive behaviors of the applications. Finally, participants shared their opinions about the Gort tool and its features.

Results: Participants generally had positive feedback regarding Gort. The participants' feedback was consolidated into categories and feature requests were organized by their corresponding request count. The results present the most prevalent feedback. Participants requested improvements in layout and separation of information. They asked for the ability to move components of the GUI around and be able to resize them. Another layout theme that emerged is that participants wanted better separation of information about what was happening in a particular activity in the AFG versus what was happening globally in the application.

Participants also requested that we allocate less screen real estate to the app descriptions and permissions. They asked for better ways to distill information, for instance, being able to sort network requests by various aspects such as time or IP address. With respect to the traversals, participants wanted a timeline to show the order that the traversal was taking place. They also wanted to know if network requests were triggered by traversal UI interactions or app code, and the frequency of the requests. Furthermore, participants asked for grouping network transmission and server information together to reduce the need to look at two areas of the GUI at once. Participants also wanted to see the use of encryption (e.g., SSL) for network requests highlighted. With respect to Android permissions, participants requested to see which permissions where used in which states. Participants rarely used the information filters included in the first version of Gort.

7.2 Second Iteration of the Gort Tool

The user study from Gort v1 informed the requirements and design for the second iteration. Furthermore, Gort v2 has the heuristic framework built-in, not just to enable privacy analysts to use a set of default heuristics to evaluate application privacy behavior, but also to enable them to define and create their own heuristics.

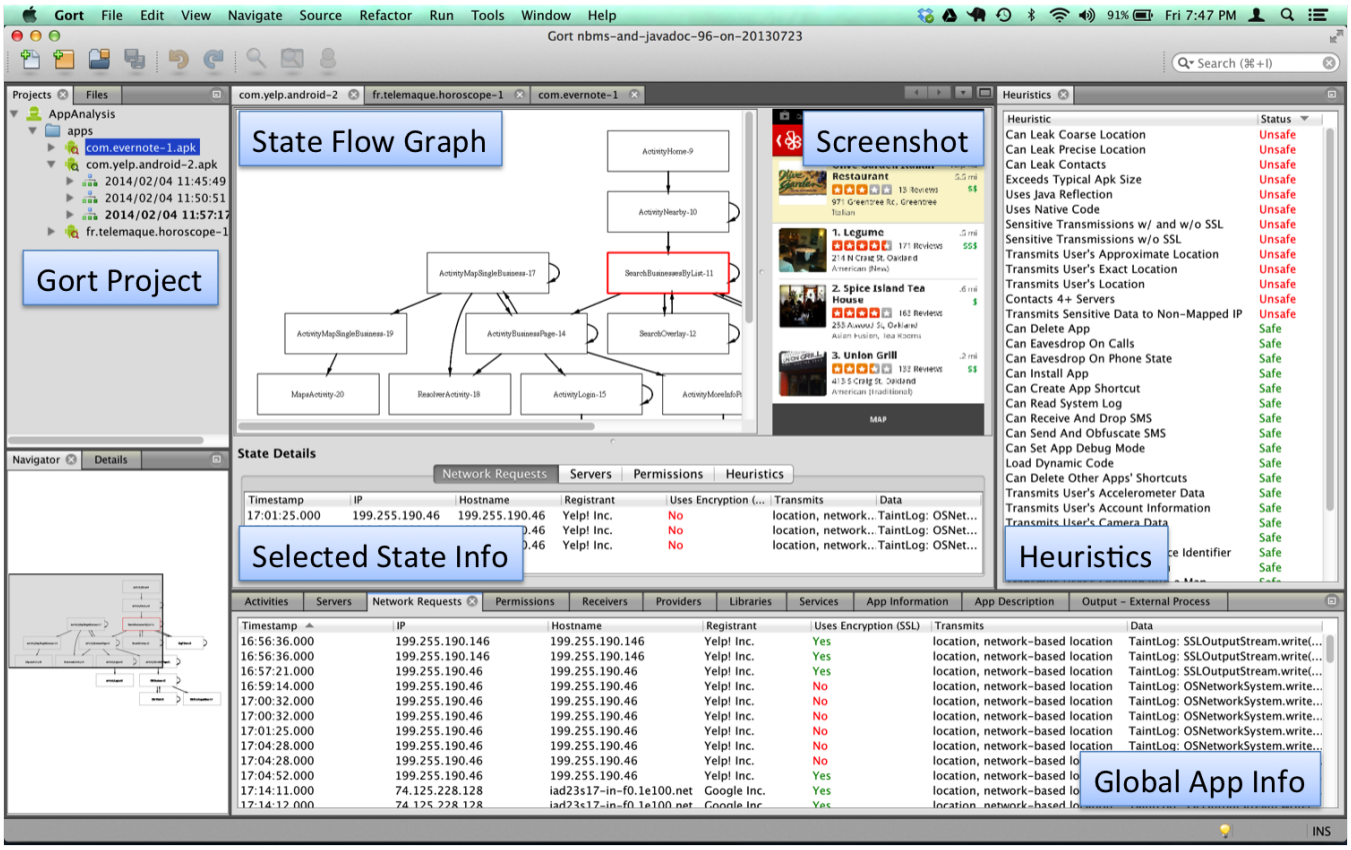


Figure 34: Gort v2 enables privacy analysts to investigate multiple apps at a time. Through the use of the heuristics, analysts can get an overview of potential privacy problems. Analysts can view different states of the app and their relationship to each other using the State Flow Graph. Gort also divides up state specific information (e.g., screenshot, network requests, permissions, heuristics) from the global app information to make it easier for analysts to pin-point specific privacy issues. All of the information panels are resizable and movable to give analysts additional freedom with respect to their ideal work environment.

7.2.1 Requirements for Gort v2

The requirements for Gort v2 include and build on the requirements from the first version (§7.1.1). Specifically, Gort v2 shares the requirements of automating application scans, exposing sensitive transmissions, and presenting behavior in the context of the application. However, Gort v2 has the following additional requirements:

Flag Potential Problems: In order to reduce the amount of repetitive and tedious interaction with scan details to analyze an app, Gort v2 should flag potential privacy problems. Supporting the heuristics framework in Gort v2, accomplishes this requirement. Furthermore, the framework must be modular, such that privacy analysts can define and create their own heuristics for evaluating and summarizing mobile app behavior.

Provide a Customizable Layout: Participants from the first user study asked for the capability to move and resize information panels. In addition to supporting this feature, Gort v2 must enable analysts to view multiple apps at a time rather than having to separately open and close the analysis for each app.

Improve Information Presentation: Based on the first user study, participants had difficulty distinguishing which information pertained to a specific state or the entire app, even though Gort v1 supported information organization through brushing and linking [15]. The second version has to better segregate information between states and the entire app. Gort v2 also has to improve the presentation of network request and server information to streamline analysis by reducing the need to interact with multiple information panels.

7.2.2 Design for Gort v2

Gort v2 was designed with the aforementioned requirements. Furthermore, Squiddy v2 was enhanced to provide more coverage for the application traversal. For the details of Squiddy v2 refer to §4.4. This section presents the design of the current version of Gort, specifically with respect to the GUI.

Figure 34 shows an overview of the second version of the GUI. Gort v2 was designed to support a customizable analysis environment, specifically all of the information panels are movable and resizable based on the analyst's preferred way of viewing application information. Gort v2 also supports creating analysis projects, allowing analysts to select apps that they would like to analyze using a project creation wizard. Analysts can add or drop new applications into the project at any time.

The top right section of the tool shows the heuristics and their outputs for both the static and dynamic heuristics. Moreover, application behavior is isolated into two sections. The top midsection of the GUI presents a *State Flow Graph* (SFG), which allows the analysts to select and view information for a single state including the network requests, servers, permissions, and heuristics. The bottom midsection of the application shows global application information. Specifically, it shows all the application core components: Activities, Services, Receivers, Providers, and Permissions for an application. It also shows the application network requests, servers, and the app information and description as available on the Google Play store. Furthermore, the visualization offers information about third-party libraries used by the app. Per feedback from the first user study, Gort v2 also combines server information with network requests to present analysts with more holistic information regarding networks, and present application information and descriptions as tabs rather than separate panels.

7.2.3 Implementation of Gort v2

I implemented Gort v2 as a NetBeans Platform Application. The underlying NetBeans Platform framework offers improved layout support, which enables the resizing and repositioning of all the information panels. Furthermore, this approach helped with porting over some of the Gort v1 Java Swing code. Gort v2 uses the Lookup[25] concept from NetBeans to offer a modular design of the heuristics for analysts to extend the built-in heuristics module, define their own heuristics module, or use a combination of the two. Gort v2 uses PostgreSQL to enable better database support in both Python and Java. The second version continues the use of the 2.3.4 version of TaintDroid as it was found to be the most stable build.

7.2.4 Evaluation of Gort v2

As in the first version of Gort, I evaluated the second version by doing a coverage analysis of the traversal algorithm as well as a user study. For the evaluation of the second iteration of the traversal algorithm, refer to §4.4.7. The evaluation of Gort v2 covered here focuses on the usability and effectiveness of the tool. For more information regarding the default heuristics built into Gort v2, refer to §5.3.3.

User Study: I performed a user study to evaluate the usability of Gort v2. The goal of the user study was to evaluate whether participants with no prior app analysis experience and expertise could answer the same type of questions analysts generally ask and answer regarding app privacy. Furthermore, I sought to evaluate the general usability of the tool, the heuristics framework, and crowd analysis.

Recruiting focused on undergraduate and master's students who had prior experience with Android. Students were recruited through posting fliers and email. All participants were compensated with $30 in cash. 14 students and one professional developer (M=8, F=7) were recruited to use Gort to analyze three Android apps. 80% of the participants were between 18 and 24 years old. All participants were recruited to have experience developing at least one app for the Android platform; however, three participants had only designed rather than developed apps for Android. The first app (WeatherBug) was used as a practice round for participants to become familiar with the tool. This section presents the results from the two other apps that participants analyzed (Horoscope and Yelp). The two apps were counterbalanced amongst the 15 participants. The tasks are referred to as t1 (Task One) and t2 (Task Two). Participants filled out a web-based survey to answer demographic questions, questions regarding the apps, and general questions regarding the Gort tool. In a follow-up interview, conducted immediately after the completion of the web-based questions, participants were asked for general feedback regarding the tool. Unless clarified otherwise, the questions referred to here are based on the web-based survey. Most sessions took approximately about an hour. All sessions were completed in less than one and a half hours. For the study protocol refer to Appendix F.

[25] `http://bits.netbeans.org/dev/javadoc/org-openide-util-lookup/overview-summary.html`

Participants were asked questions to determine their expertise level. They were asked about their comfort with a series of tasks on a 5-point Likert scale. The medians for comfort with the following tasks were: sending emails (Strongly Agree), working with spreadsheets (Agree), developing in Java (Strongly Agree), and understanding the Android activity life-cycle (Agree). However, the median for understanding the Open Systems Interconnection (OSI) model was Disagree, and writing a buffer overflow exploit was Neutral. As such, the participants had a good understanding of Android, but were intentionally selected not be experts in computer networking or security. As discussed in §1.3.2, the goal with Gort is to support people who have basic knowledge and skills for app analysis, but may not necessarily be experts.

The study methodology was the following. For each app, participants were presented with a set of questions typically answered by analysts when evaluating app privacy (e.g., Does the app send unencrypted messages? Does the app transmit the user's location? Does the app use the device's camera?). For a complete list of the questions, refer to Appendix G. The logic here was that if students without mobile privacy training can correctly answer app privacy questions in a short amount of time, the tool should help privacy analysts who have even more training. Furthermore, as Gort is the first interactive tool to give a comprehensive view of Android apps, there is no baseline for comparison. However, by conducting a user study, the usability of the tool can still be studied.

Participants were asked 26 questions in a period of 15 minute for each app. Four questions pertained to the name and description of the app, and prior experience with it. The other 22 questions are typical questions that analysts would answer with respect to app privacy, with randomized order for each participant. Participants had to read and think aloud about the questions. They were also able to skip questions. Finally, participants were asked about their general opinion and feedback regarding the tool.

Results: This section reports the results of tasks t1 and t2. All participants successfully answered the four required questions regarding name, description, and prior experience with the apps. The scores for the 22 optional questions were calculated for each participant, where correctly answering a question would result in 1 point for a total of 22 points. Participants performed well on both t1 (μ=18.6, Mdn=20, σ=3.44) and t2 (μ=20.3, Mdn=21, σ=1.63). Considering that participants correctly answered over 84% of the questions for t1 and over 92% for t2, the Gort tool was successful in enabling participants to correctly answer the same type of questions that experts would asks regarding apps' privacy aspects.

It was expected to observe a learning effect as participants became more oriented with the tool. A learning effect was indeed observed as participants got more experience using the tool from t1 to t2. Specifically, a Wilcoxon signed-rank test was computed to check for the learning effect. The Wilcoxon test showed the difference in the scores for t1 and t2 to be statistically significant (p=0.0115). Furthermore, participants generally used more time during t1 (μ=681s, Mdn=677s, σ=129s) than t2 (μ=524s, Mdn=520s, σ=114s). The time difference was significant using a paired t-test ($t(14)=3.95$, p=0.0014).

Participants found that it was easy to find what sensitive information was sent for both t1 (μ=3.93, Mdn=4, σ=1.10) and t2 (μ=4.26, Mdn=4, σ=0.88). Furthermore, participants felt confident in their responses for t1 (μ=4.13, Mdn=4, σ=0.64) and t2 (μ=4.26, Mdn=4, σ=0.69). The differences in ease of finding sensitive information and confidence in responses over the two tasks were not significant using a Wilcoxon signed-rank test. A number of participants had used some apps prior to the study. To understand whether having used an app previously introduced a bias in the results, significance tests were performed for the number of correct answers, ease of finding information, and confidence in the results. A Wilcoxon rank-sum test was performed for each of the three cases. There were no significant differences for cases when participants had used an app prior to the study. As a result, having used an app previously did not introduce a bias in how a participant answered app privacy questions.

Following t1 and t2, participants were asked general questions regarding the tool on a 5-point Likert scale (5 Strongly Agree). Participants found that the tool offered enough details to answer the questions about app privacy (Mdn=4) and that the user interface was easy to use and follow (Mdn=4). Participants would consider using this tool again to analyze app privacy (Mdn=5) and found the heuristics useful (Mdn=5). In free-form responses and the follow-up interview, participants asked for reverse-lookups from heuristics and network requests to see in which states they occurred. Gort already supports showing which heuristics and network requests correspond to specific states. Furthermore, participants asked for the heuristics to be grouped by type, for instance, anything concerning the camera or anything concerning location. They would still like the feature of sorting heuristics based on which ones marked the app as unsafe. Furthermore, participants mentioned that they would find it useful if the tool generated a report for each app.

7.3 Summary

As a part of this thesis, I developed two versions of the Gort GUI tool. The Gort GUI offers an interactive tool for privacy analysts to inspect Android apps. I performed coverage analysis of the two corresponding traversal algorithms and also conducted user studies to evaluate each of the tools.

The preliminary version, Gort v1, streamlines the end-to-end analysis process. I built the first version to understand the space, learn about challenges of analyzing app privacy, and verify the feasibility of the tool. Gort v1 offers analysts an interactive flow graph of apps along with the associated network requests, servers, and TaintDroid logs. I performed a user study with 12 experts to evaluate Gort GUI v1. While the study helped understand the space and highlighted many challenges, there were two fundamental takeaways from building and evaluating Gort v1. First, app traversal is extremely challenging. The challenges and potential solutions to mitigate them are covered in §4. Second, it was observed that some frequent and routine tasks to analyze apps required participants to inspect app network requests and permissions

in-depth. To reduce the amount of in-depth analysis required for detecting app privacy problems, I devised a heuristics framework, which is built into Gort v2.

The second version of Gort incorporates solutions to improve app traversal, a heuristics framework to detect potential app privacy problems, and crowd analysis to understand app context. The updates to the traversal algorithm as well as the improvement in app coverage are presented in §4. For details on the heuristics and the heuristics framework, refer to §5. The crowd analysis approach and results are covered in §6. Gort v2 was evaluated through a usability study with 15 participants. The recruiting process intentionally focused on students who had experience developing at least one Android app, but did not have expertise in computer privacy or security. The focus of the study was to find out whether the tool was effective in enabling people who do not have expertise in privacy analysis to answer the same set of questions that privacy analysts would ask and answer regarding apps. The participants analyzed two apps with Gort v2 and answered questions in a 15 minutes time period for each app. Participants were able to correctly answer over 84% and over 92% of the questions for the first and second task, respectively. A learning effect was observed from the first task to the second. Participants answered more questions correctly in a shorter period of time during the second task. Participants found the tool and the heuristics useful. They indicated that the tool offered enough details to answer the questions and that they would consider using it again to analyze app privacy.

The usability study of Gort v2 had several important takeaways. First, Gort enabled the participants to effectively analyze mobile apps and answer the same type of questions that a privacy analysts would ask about app privacy. Second, the participants found the heuristics extremely useful in answering the questions, but there is still room to improve the presentation of heuristics. The presentation could be enhanced by using a hierarchical grouping of heuristics and offering search functionality (refer to §5). Third, participants requested for Gort to generate a summary report after analyzing each app. Future work can consider improvements in heuristic presentation and summary report generation as potential directions to enhance Gort.

8 Conclusion and Future Work

This chapter concludes the thesis with a summary and presents the implications of this work to provide insights for future work and guide the design of future systems and tools for evaluating mobile app privacy.

8.1 Thesis Summary

This thesis offers fundamental contributions to the field of mobile privacy and crowdsourcing in four distinct ways:

First, Gort offers a heuristics framework that helps privacy analysts find potential app privacy problems quickly. The heuristics provide a set of red flags for potentially suspicious app behavior. These red flags give analysts a high-level overview of the app's potential problems. Prior to this work, analysts did not have an easy way of using cues or red flags to view app privacy problems. The heuristic framework is built into Gort with a set of default heuristics. The author compiled a set of heuristics by studying prior work, manually inspecting over 40 popular Android apps for suspicious behaviors, and eliciting heuristics by interviewing 12 experts. This process resulted in over 100 heuristics, 35 of which are built into Gort as default heuristics. The heuristics framework allows analysts to choose from Gort's default heuristics or create their own custom heuristics. Apart from the heuristic framework and the built-in heuristics implemented, the process followed during the interviews to elicit heuristics can be used as a guide for other researchers interested in eliciting heuristics for other purposes.

Second, the thesis presents an automated approach to interact with mobile apps while instrumenting them to find potential privacy problems. The automated approach is comprised of Gort's app traversal algorithm, Squiddy. Squiddy automatically discovers UI elements in an app's GUI and interacts with them to explore various states of the app. Using this approach, app analysis does not have to rely on a person to manually trigger app behavior or script the interactions to trigger app behavior. This thesis presents the challenges faced while building Squiddy, offers several approaches to overcome the most prohibiting challenges, and provides insights for future work on how to addresses the remaining ones.

Third, this thesis presents a fundamentally new approach to analyze app behavior with respect to app context and semantics. While existing app analysis tools offer techniques to detect potential app privacy leaks, they cannot present whether these privacy leaks are legitimate and beneficial to the user. Through obtaining the context and semantics of apps using crowdsourcing, Gort can request crowd opinions regarding app behavior by providing the context and semantics to the crowd as well as the privacy intrusive app behaviors. Using this approach, Gort raises flags for privacy intrusive app behavior that is not deemed desirable or beneficial by crowd workers. This thesis also offers the *extraction-verification-justification*

crowdsourcing pattern that can be used by others to evaluate app privacy and perhaps extended to evaluate other aspects of apps or software in general, e.g., usability, design.

Fourth, the Gort System and the interactive GUI tool present a working implementation of the ideas and approaches presented in this thesis. The author has built two different versions of Gort, namely, Gort v1 and v2. Both versions were evaluated with user studies with 12 and 15 users, respectively. The first version required approximately 10,000 source lines of code, written by the author. The latest implementation (v2) consists of over 35,000 source lines of code by the author, which has required immense time, effort, dedication, and knowhow. Nevertheless, a working system is the only way to show that the ideas proposed in this thesis really work, not just by themselves, but also when integrated together. The implementation provided presents the first semi-automated system to analyze software, specifically mobile apps, with respect to application context and semantics.

8.2 Discussion

In this section, I discuss the implications of the work presented in this thesis, which should provide insights into designing better systems and tools for evaluating app privacy.

8.2.1 App Traversal is Challenging

Traversing mobile apps is surprisingly challenging. To name a few of the many challenges present, apps may exhibit non-deterministic behavior, require complex interactions to trigger behavior, request user authentication, and even change the underlying system environment while being traversed. Support for fast and high-coverage traversal algorithms could enable a range of app analysis with respect to privacy, security, usability, and more. Current traversal algorithms all work around platform challenges to provide reasonable while not highly efficient app traversals. Mobile app platforms or third-parties may be interested in providing better support for app traversals to enable much needed automated analysis and evaluation of apps. They may also explore how to combine the results of several traversals to better identify whether app behavior is deterministic, sporadic, or random and offer better statistics regarding the frequency of event occurrences.

Several challenges can be easily mitigated through better app traversal support by platform owners. For instance, similar to Gort's approach, the Android platform may expose when the virtual keyboard is visible on the screen and hiding UI components underneath it. The platform can also report when the screen has finished rendering to inform traversal algorithms when to read the screen and prevent them from reading the screen multiple times. Offering more detailed information about callback functions triggered by UI components would also be a welcome addition. There are a number of potential improvements that

would take more effort. One example would be to provide high-level information about which UI elements would result in activity change or state transitions. Another would be to create an *isolate-app* mode that supports the analysis of a single app and minimizes perturbations by other apps and system components. §4.3 provides many other traversal hurdles that can be addressed by platform owners.

8.2.2 Privacy Heuristics Have Great Potential

Based on the user study results from Gort v2 (§7.2.4), participants were capable of correctly answering questions commonly answered by privacy analysts about the privacy aspects of using an app. The participants were intentionally recruited to be students without expertise in computer privacy and security. However, they were still able to answer more than 90% of the questions correctly.

The heuristics are a core part of Gort that enable even non-experts to evaluate mobile app privacy. However, as discussed in the evaluation of heuristics (§5.3.3), there were cases when participants would rely too much on heuristics. For instance, they would assume that an app does not use the smartphone's camera, if it did not *transmit user's camera data*. It is recommended for privacy analysts to go through training to clearly understand the meaning of heuristics. Heuristics designers should take steps to clearly define heuristics and what privacy problem each heuristic targets. Moreover, heuristics that do not mark an app as unsafe may have been limited by the intermediary results from static analysis or the app traversal. As such, providing a confidence value for the output of heuristics would be beneficial.

The presentation of heuristics should also be explored. In the current implementation, Gort's heuristics are presented as a table that may be sorted by the heuristics' names or output values. If the number of heuristics increases such that they are no longer manageable through a table, better presentation of heuristics would be necessary. Some directions in this space are to make the heuristics searchable and also to provide a hierarchy of heuristics. For instance, all heuristics related to the smartphone's camera could be grouped together, e.g., uses camera, transmits camera data, takes photos, takes videos.

8.2.3 It Takes an Ecosystem

This thesis studies the mobile app ecosystem with respect to privacy analysis and supporting analysts with an interactive tool for app privacy analysis. However, the mobile ecosystem is vast and there remain many other aspects that can be explored. Truly offering privacy-aware and secure apps to end-users will require effort on multiple fronts. Platform maintainers need to provide developers with better tools and the incentive to make their apps respect user privacy. They also need to make app privacy more transparent to end-users and enable them to make informed decisions. Developers should educate themselves about app privacy and learn about the implications of using third-party advertisement networks and analytics firms

with respect to their users' privacy. End-users need to better understand mobile privacy and the implications of using privacy-intrusive apps. Journalists and third-party analysts can bring more app privacy problems to the public's knowledge. Entities such as the FTC and government officials need to pave the path for better mobile privacy by updating regulation and policy to keep up with technological trends that potentially threaten user privacy. We as researchers can study the needs of these entities and individuals and create systems, tools, and approaches to address their needs in a way that protects consumers' privacy.

8.3 Future Work

As the work presented in this thesis builds on a variety of research domains and involves an entire end-to-end system, it offers for a variety of exciting opportunities for future work.

8.3.1 Enabling App Comparison and Scoring

The Gort tool enables multiple apps to be traversed and the results to be viewed in the same GUI environment side-by-side. However, there is much more that can be done with respect to app comparison. Future analysis tools can extend Gort by offering better mobile app comparison features. For instance, through offering tools for organizing apps (e.g., buckets for privacy-intrusive, privacy-preserving, malicious, and to-be-determined apps) and annotations to record results and notes from the analyst's work. Furthermore, future tools could explore how automated processes can be used to produce a privacy score for an application that can be easily understood by end-users. The privacy score should focus on helping end-users make informed decisions with regard to installing and using apps. Researchers may also study how to support privacy analysts, who may be interested in scoring a set of applications based on their expertise, with better scoring tools and approaches.

8.3.2 Supporting Collaborative Analysis

Considering the scale and growth of the app markets, and privacy evaluation frequently being a subjective process, mobile app analysis could benefit from collaboration. Supporting collaboration could be approached in a variety of ways. For instance, app market maintainers could traverse apps and provide the results and play-backs to privacy analysts and developers through the cloud. Also, a repository of analyzed apps could help bring together the efforts of many analysts, similar to how entities such as Stackoverflow[26] combine the efforts of many developers. Since a Gort project is an isolated collection of apps, data, and

[26] `http://stackoverflow.com`

analysis, Gort could provide a wizard to import and export projects to encourage collaboration through sharing analysis results.

8.3.3 Offering Privacy-Aware Development Tools

Developers can use better tools and frameworks to create privacy-preserving apps. For example, Caché is another work by the author, that explores mobile privacy from a development perspective [10]. Researchers may be interested in extending Gort or adopting some of its features in development environments to better enable developers with respect to mobile privacy. For instance, development tools could inform developers about how various third-party ad network and analytics libraries behave. For lesser known libraries, the tools can automatically scan the app using static and dynamic analysis techniques to offer the developer information about how their app behaves or privacy heuristics that their apps would not pass as a result of specific libraries. Using the crowd approach, developers could get information about what potential users would think of their app before sending the app to the market. Developer tools are in need of drastic improvement. There is potential for great impact in this space.

8.3.4 Optimizing and Extending Crowd Analysis

The crowdsourcing approach proposed in this thesis and the *extraction-verification-justification* pattern offer one solution to evaluate app privacy with respect to context and semantics of apps. These efforts can be extended in several ways. Here, I propose four potential directions. First, researchers can explore the use of the proposed approach and crowdsourcing pattern to evaluate other aspects of mobile apps, for instance, usability or design. Second, researchers can extend the approach for software beyond apps (e.g., desktop applications) and study whether the approach is extendable to complex applications beyond mobile computing. Third, the proposed approach can be optimized to reduce the reliance on the crowd by automating more of the system or with respect to the amount of monetary compensation offered to workers. Fourth, researchers can attempt to come up with different patterns for mobile app evaluation or perhaps crowdsourcing in general to extend the arsenal of patterns available to answer complex and even technical questions using non-technically oriented crowdsourcing platforms.

8.3.5 Expanding Beyond Android

The work presented in this thesis has focused on the Android mobile platform; however, many of the ideas and concepts presented here may be adopted for other platforms (e.g., iOS) and other application markets

(e.g., app markets for cars, homes, TVs, and web browsers). Technology companies such as Google, Apple, and Microsoft have long carved out their stakes in the web browser market. However, in recent years, these companies have also moved into users' living rooms and cars through technologies such as Google Chromecast, Apple TV, and Microsoft XBox and SYNC. There is tremendous opportunity to provide users with compelling applications and services on these platforms; however, there is also tremendous potential for exploiting users. The privacy implications of the new technologies should be studied to reduce the increasing gap between rapidly growing technologies and the required privacy protection mechanisms to support them.

8.4 Closing Remarks

Privacy is most valuable and hardest to attain once lost. If we do not build privacy into the technologies that we create today, we may inadvertently deprive future generations of a basic human right. This basic human right is not the right to one's private information. Rather, it is the right to one's autonomy. In a world where one's every step is tracked, and every communication is recorded, one may have to think twice before taking another step and uttering another word.

References

[1] Yuvraj Agarwal and Malcolm Hall. ProtectMyPrivacy: Detecting and Mitigating Privacy Leaks on iOS Devices Using Crowdsourcing. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '13, pages 97–110, New York, NY, USA, 2013. ACM.

[2] Luis Ahn, Manuel Blum, Nicholas J. Hopper, and John Langford. CAPTCHA: Using Hard AI Problems for Security. In Eli Biham, editor, *Advances in Cryptology - EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 294–311. Springer Berlin Heidelberg, 2003.

[3] Chloe Albanesius. Congress Asks Apple for Details About Address Book Privacy. *PCMAG*, Feb 2012. http://www.pcmag.com/article/print/294178.

[4] Alasdair Allan and Pete Warden. Got an iPhone or 3G iPad? Apple is recording your moves. *O'Reilly Radar*, Apr 2011. http://radar.oreilly.com/2011/04/apple-location-tracking.html.

[5] Amazon.com Inc. Amazon Appstore for Android. http://www.amazon.com/mobile-apps/b?node=2350149011.

[6] Amazon.com Inc. Amazon Mechanical Turk. https://www.mturk.com/mturk/welcome.

[7] Shahriyar Amini, A.J. Brush, John Krumm, Jaime Teevan, and Amy Karlson. Trajectory-aware Mobile Search. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 2561–2564, New York, NY, USA, 2012. ACM.

[8] Shahriyar Amini and Yang Li. CrowdLearner: Rapidly Creating Mobile Recognizers Using Crowdsourcing. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology*, UIST '13, pages 163–172, New York, NY, USA, 2013. ACM.

[9] Shahriyar Amini, Jialiu Lin, Jason I. Hong, Janne Lindqvist, and Joy Zhang. Towards Scalable Evaluation of Mobile Applications through Crowdsourcing and Automation. Technical Report CMU-CyLab-12-006, CMU Cylab, 2012.

[10] Shahriyar Amini, Janne Lindqvist, Jason Hong, Jialiu Lin, Eran Toch, and Norman Sadeh. Caché: Caching Location-enhanced Content to Improve User Privacy. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, MobiSys '11, pages 197–210, New York, NY, USA, 2011. ACM.

[11] AppBrain. Number of available Android applications. Feb 2014. http://www.appbrain.com/stats/number-of-android-apps.

[12] Apple Inc. About the iOS Technologies. Sep 2013. https://developer.apple.com/library/ios/documentation/miscellaneous/conceptual/iphoneostechoverview/Introduction/Introduction.html.

[13] Kathy Wain Yee Au, Yi Fan Zhou, Zhen Huang, Phillipa Gill, and David Lie. Short Paper: A Look at Smartphone Permission Models. In *Proceedings of the 1st ACM Workshop on Security and*

Privacy in Smartphones and Mobile Devices, SPSM '11, pages 63–68, New York, NY, USA, 2011. ACM.

[14] Tuomas Aura, Janne Lindqvist, Michael Roe, and Anish Mohammed. Chattering Laptops. In *Proceedings of the 8th international symposium on Privacy Enhancing Technologies*, PETS '08, pages 167–186, Berlin, Heidelberg, 2008. Springer-Verlag.

[15] Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[16] Rebecca Balebako, Jaeyeon Jung, Wei Lu, Lorrie Faith Cranor, and Carolyn Nguyen. "Little Brothers Watching You": Raising Awareness of Data Leaks on Smartphones. In *Proceedings of the Ninth Symposium on Usable Privacy and Security*, SOUPS '13, pages 12:1–12:11, New York, NY, USA, 2013. ACM.

[17] Michel Balinski and Rida Laraki. *Majority Judgment: Measuring, Ranking, and Electing*. The MIT Press, 2011.

[18] BBC News. Social apps 'harvest smartphone contacts'. *BBC News*, February 2012. `http://www.bbc.co.uk/news/technology-17051910?print=true`.

[19] Lutz Bendlin. iGuidance 4 NA review. Oct 2007. `http://www.pocketgpsworld.com/iguidance4.php`.

[20] Alastair R. Beresford, Andrew Rice, Nicholas Skehin, and Ripduman Sohan. MockDroid: trading privacy for application functionality on smartphones. In *Proceedings of HotMobile 2011*. ACM, 2011.

[21] Michael S. Bernstein, Greg Little, Robert C. Miller, Björn Hartmann, Mark S. Ackerman, David R. Karger, David Crowell, and Katrina Panovich. Soylent: a word processor with a crowd inside. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*, UIST '10, pages 313–322, New York, NY, USA, 2010. ACM.

[22] Nick Bilton. Disruptions: So Many Apologies, So Much Data Mining. *The New York Times Blogs*, February 2012. `http://bits.blogs.nytimes.com/2012/02/12/disruptions-so-many-apologies-so-much-data-mining/`.

[23] Stephanie Blanchard. The Good and Bad of BYOD. *Mobile Enterprise*, June 2013. `http://mobileenterprise.edgl.com/news/The-Good-and-Bad-of-BYOD86825`.

[24] Theodore Book, Adam Pridgen, and Dan S. Wallach. Longitudinal Analysis of Android Ad Library Permissions. *CoRR*, abs/1303.0857, 2013.

[25] Sven Bugiel, Lucas Davi, Alexandra Dmitrienko, Thomas Fischer, and Ahmad-Reza Sadeghi. XManDroid: A New Android Evolution to Mitigate Privilege Escalation Attacks. Technical Report TR-2011-04, Technische Universität Darmstadt, April 2011.

[26] Sven Bugiel, Lucas Davi, Alexandra Dmitrienko, Stephan Heuser, Ahmad-Reza Sadeghi, and Bhargava Shastry. Practical and Lightweight Domain Isolation on Android. In *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '11, pages 51–62, New York, NY, USA, 2011. ACM.

[27] Iker Burguera, Urko Zurutuza, and Simin Nadjm-Tehrani. Crowdroid: Behavior-based Malware Detection System for Android. In *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '11, pages 15–26, New York, NY, USA, 2011. ACM.

[28] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka, and Tom M. Mitchell. Toward an Architecture for Never-Ending Language Learning. In *In AAAI*, 2010.

[29] Lorenzo Cavallaro, Prateek Saxena, and R. Sekar. On the Limits of Information Flow Techniques for Malware Analysis and Containment. In *Proceedings of the 5th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, DIMVA '08, pages 143–163, Berlin, Heidelberg, 2008. Springer-Verlag.

[30] Avik Chaudhuri. Language-based Security on Android. In *Proceedings of the ACM SIGPLAN Fourth Workshop on Programming Languages and Analysis for Security*, PLAS '09, pages 1–7, New York, NY, USA, 2009. ACM.

[31] Erika Chin, Adrienne Porter Felt, Kate Greenwood, and David Wagner. Analyzing Inter-application Communication in Android. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, MobiSys '11, pages 239–252, New York, NY, USA, 2011. ACM.

[32] Erika Chin, Adrienne Porter Felt, Vyas Sekar, and David Wagner. Measuring User Confidence in Smartphone Security and Privacy. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, SOUPS '12, pages 1:1–1:16, New York, NY, USA, 2012. ACM.

[33] Edmund M. Clarke, William Klieber, Miloš Nováček, and Paolo Zuliani. Model Checking and the State Explosion Problem. In Bertrand Meyer and Martin Nordio, editors, *Tools for Practical Software Verification*, volume 7682 of *Lecture Notes in Computer Science*, pages 1–30. Springer Berlin Heidelberg, 2012.

[34] Sunny Consolvo, Jaeyeon Jung, Ben Greenstein, Pauline Powledge, Gabriel Maganis, and Daniel Avrahami. The Wi-Fi Privacy Ticker: Improving Awareness & Control of Personal Information Exposure on Wi-Fi. In *Proceedings of the 12th ACM International Conference on Ubiquitous Computing*, Ubicomp '10, pages 321–330, New York, NY, USA, 2010. ACM.

[35] Mauro Conti, VuThienNga Nguyen, and Bruno Crispo. CRePE: Context-Related Policy Enforcement for Android. In Mike Burmester, Gene Tsudik, Spyros Magliveras, and Ivana Ili, editors, *Information Security*, volume 6531 of *Lecture Notes in Computer Science*, pages 331–345. Springer Berlin Heidelberg, 2011.

[36] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[37] Anthony Desnos and Geoffroy Gueguen. Android: From Reversing to Decompilation. *Androguard*, Dec 2012.

[38] Michael Dietz, Shashi Shekhar, Yuliy Pisetsky, Anhei Shu, and Dan S. Wallach. Quire: Lightweight Provenance for Smart Phone Operating Systems. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, pages 23–23, Berkeley, CA, USA, 2011. USENIX Association.

[39] Julie S. Downs, Mandy B. Holbrook, Steve Sheng, and Lorrie Faith Cranor. Are Your Participants Gaming the System?: Screening Mechanical Turk Workers. In *Proceedings of the SIGCHI Confer-*

ence on Human Factors in Computing Systems, CHI '10, pages 2399–2402, New York, NY, USA, 2010. ACM.

[40] Manuel Egele, Christopher Kruegel, Engin Kirda, and Giovanni Vigna. PiOS : Detecting privacy leaks in iOS applications. In *NDSS '11*, 2011.

[41] Serge Egelman, AdriennePorter Felt, and David Wagner. Choice Architecture and Smartphone Privacy: There's a Price for That. In Rainer Bhme, editor, *The Economics of Information Security and Privacy*, pages 211–236. Springer Berlin Heidelberg, 2013.

[42] William Enck. Defending Users against Smartphone Apps: Techniques and Future Directions. In Sushil Jajodia and Chandan Mazumdar, editors, *Information Systems Security*, volume 7093 of *Lecture Notes in Computer Science*, pages 49–70. Springer Berlin Heidelberg, 2011.

[43] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association.

[44] William Enck, Damien Octeau, Patrick McDaniel, and Swarat Chaudhuri. A Study of Android Application Security. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, pages 21–21, Berkeley, CA, USA, 2011. USENIX Association.

[45] William Enck, Machigar Ongtang, and Patrick McDaniel. On Lightweight Mobile Phone Application Certification. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 235–245, New York, NY, USA, 2009. ACM.

[46] Sascha Fahl, Marian Harbach, Thomas Muders, Lars Baumgärtner, Bernd Freisleben, and Matthew Smith. Why Eve and Mallory Love Android: An Analysis of Android SSL (in)Security. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 50–61, New York, NY, USA, 2012. ACM.

[47] Federal Trade Commission. Android Flashlight App Developer Settles FTC Charges It Deceived Consumers. *FTC Press Releases*, December 2013.

[48] Federal Trade Commission. Path Social Networking App Settles FTC Charges it Deceived Consumers and Improperly Collected Personal Information from Users' Mobile Address Books. *FTC Press Releases*, February 2013.

[49] Joseph Feiman and Dionisio Zumerle. Path 2 uploads your address book, but says it's to 'match friends' and will be opt-in soon. *TNW*, February 2012. `http://thenextweb.com/apps/2012/02/07/path-2-uploads-your-address-book-but-says-that-its-for-friend-matching-and-will-be-opt-in-soon`.

[50] Joseph Feiman and Dionisio Zumerle. Technology Overview: Mobile Application Security Testing for BYOD Strategies. *Gartner*, August 2013. `http://www.gartner.com/technology/reprints.do?id=1-1KCI97T&ct=130913&st=sb#h-d2e335`.

[51] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. Android Permissions Demystified. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 627–638, New York, NY, USA, 2011. ACM.

[52] Adrienne Porter Felt, Serge Egelman, Matthew Finifter, Devdatta Akhawe, and David Wagner. How to Ask for Permission. In *Proceedings of the 7th USENIX Conference on Hot Topics in Security*, HotSec'12, pages 7–7, Berkeley, CA, USA, 2012. USENIX Association.

[53] Adrienne Porter Felt, Kate Greenwood, and David Wagner. The Effectiveness of Application Permissions. In *Proceedings of the 2Nd USENIX Conference on Web Application Development*, WebApps'11, pages 7–7, Berkeley, CA, USA, 2011. USENIX Association.

[54] Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin, and David Wagner. Android Permissions: User Attention, Comprehension, and Behavior. Technical Report UCB/EECS-2012-26, University of California at Berkeley, 2012.

[55] Adrienne Porter Felt, Helen J. Wang, Alexander Moshchuk, Steven Hanna, and Erika Chin. Permission Re-delegation: Attacks and Defenses. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, pages 22–22, Berkeley, CA, USA, 2011. USENIX Association.

[56] Mario Frank, Ben Dong, Adrienne Porter Felt, and Dawn Song. Mining Permission Request Patterns from Android and Facebook Applications. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*, pages 870–875, Dec 2012.

[57] Adam P. Fuchs, Avik Chaudhuri, and Jeffrey S. Foster. SCanDroid: Automated Security Certification of Android Applications.

[58] Christopher Gates, Ninghui Li, Hao Peng, Bhaskar Sarma, Yuan Qi, Rahul Potharaju, Cristina Nita-Rotaru, and Ian Molloy. Generating Summary Risk Scores for Mobile Applications. *Dependable and Secure Computing, IEEE Transactions on*, PP(99):1–1, 2014.

[59] Alison Gendar. How Cell Phone Helped Cops Nail Key Murder Suspect. Secret 'Pings' That Gave Bouncer Away. *Dailynews*, Mar 2006. `http://www.nydailynews.com/archives/news/cell-phone-helped-cops-nail-key-murder-suspect-secret-pings-gave-bouncer-article-1.599672`.

[60] Clint Gibler, Jonathan Crussell, Jeremy Erickson, and Hao Chen. AndroidLeaks: Automatically Detecting Potential Privacy Leaks in Android Applications on a Large Scale. In *Proceedings of the 5th International Conference on Trust and Trustworthy Computing*, TRUST'12, pages 291–307, Berlin, Heidelberg, 2012. Springer-Verlag.

[61] Google Inc. Android. `http://www.android.com`.

[62] Google Inc. Android Debug Bridge. `http://developer.android.com/tools/help/adb.html`.

[63] Google Inc. Android, the world's most popular mobile platform. `http://developer.android.com/about/index.html`.

[64] Google Inc. Application Fundamentals. `http://developer.android.com/guide/components/fundamentals.html`.

[65] Google Inc. Google App Engine. `http://developers.google.com/appengine/`.

[66] Google Inc. Introducing ART. `http://source.android.com/devices/tech/dalvik/art.html`.

[67] Google Inc. Security Issue. Mar 2011. http://market.android.com/support/bin/answer.py?answer=1207928.

[68] Google Inc. Visibility for Your Apps. Mar 2011. http://developer.android.com/distribute/googleplay/about/visibility.html.

[69] Google Inc. Security and Permissions. *Android Developers*, Feb 2012. http://developer.android.com/guide/topics/security/security.html.

[70] Michael Grace, Yajin Zhou, Qiang Zhang, Shihong Zou, and Xuxian Jiang. RiskRanker: Scalable and Accurate Zero-day Android Malware Detection. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, MobiSys '12, pages 281–294, New York, NY, USA, 2012. ACM.

[71] Graphviz. Graph Visualization Software. http://graphviz.org/.

[72] Grappa. AT&T Labs Research - Grappa. http://www2.research.att.com/~john/Grappa/.

[73] Sam Grobart. The Facebook Scare That Wasn't. *The New York Times Blog*, August 2011. http://gadgetwise.blogs.nytimes.com/2011/08/10/the-facebook-scare-that-wasnt/.

[74] John A. Gubner. *Probability and Random Processes for Electrical and Computer Engineers*. Cambridge University Press, 2006.

[75] Jeffrey Heer and Michael Bostock. Crowdsourcing Graphical Perception: Using Mechanical Turk to Assess Visualization Design. In *Proceedings of the 28th International Conference on Human Factors in Computing Systems*, CHI '10, pages 203–212, New York, NY, USA, 2010. ACM.

[76] Kurtis Heimerl, Brian Gawalt, Kuang Chen, Tapan Parikh, and Björn Hartmann. CommunitySourcing: Engaging Local Crowds to Perform Expert Work via Physical Kiosks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 1539–1548, New York, NY, USA, 2012. ACM.

[77] D. Richard Hipp. SQLite. http://www.sqlite.org.

[78] Daniel Hoffman. Exposing Your Personal Information - There's An App for That. *Juniper Networks Blog*, Oct 2012. http://forums.juniper.net/t5/Security-Mobility-Now/Exposing-Your-Personal-Information-There-s-An-App-for-That/ba-p/166058.

[79] Peter Hornyack, Seungyeop Han, Jaeyeon Jung, Stuart Schechter, and David Wetherall. "These Aren't the Droids You're Looking For": Retrofitting Android to Protect Data from Imperious Applications. In *Proceedings of the 18th ACM conference on Computer and Communications Security*, CCS '11, pages 639–652, New York, NY, USA, 2011. ACM.

[80] HP Labs. Tesseract. https://code.google.com/p/tesseract-ocr/.

[81] Troy Hunt. Secret iOS business; what you don't know about your apps. *Troy Hunt's Blog*, Oct 2011.

[82] Nathan Ingraham. Apple announces 1 million apps in the App Store, more than 1 billion songs played on iTunes radio. Oct 2013. http://www.theverge.com/2013/10/22/4866302/apple-announces-1-million-apps-in-the-app-store.

[83] Panagiotis G. Ipeirotis, Foster Provost, and Jing Wang. Quality Management on Amazon Mechanical Turk. In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, HCOMP '10, pages 64–67, New York, NY, USA, 2010. ACM.

[84] Jinseong Jeon, Kristopher K Micinski, Jeffrey A Vaughan, Ari Fogel, Nikhilesh Reddy, Jeffrey S Foster, and Todd Millstein. Dr. Android and Mr. Hide: Fine-grained Permissions in Android Applications. In *Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '12, pages 3–14. ACM, 2012.

[85] Jaeyeon Jung, Seungyeop Han, and David Wetherall. Short Paper: Enhancing Mobile Application Permissions with Runtime Feedback and Constraints. In *Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '12, pages 45–50, New York, NY, USA, 2012. ACM.

[86] Jaeyeon Jung, Anmol Sheth, Ben Greenstein, David Wetherall, Gabriel Maganis, and Tadayoshi Kohno. Privacy Oracle: a System for Finding Application Leaks with Black Box Differential Testing. In *Proceedings of the 15th ACM Conference on Computer and Communications Security*, CCS '08, pages 279–288, New York, NY, USA, 2008. ACM.

[87] Patrick Gage Kelley, Sunny Consolvo, Lorrie Faith Cranor, Jaeyeon Jung, Norman Sadeh, and David Wetherall. A Conundrum of Permissions: Installing Applications on an Android Smartphone. In *Proceedings of the 16th International Conference on Financial Cryptography and Data Security*, FC'12, pages 68–79, Berlin, Heidelberg, 2012. Springer-Verlag.

[88] Patrick Gage Kelley, Lorrie Faith Cranor, and Norman Sadeh. Privacy As Part of the App Decision-making Process. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 3393–3402, New York, NY, USA, 2013. ACM.

[89] Aniket Kittur, Ed H. Chi, and Bongwon Suh. Crowdsourcing user studies with Mechanical Turk. In *Proceedings of the Twenty-sixth Annual SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, pages 453–456, New York, NY, USA, 2008. ACM.

[90] Aniket Kittur and Robert E. Kraut. Harnessing the Wisdom of Crowds in Wikipedia: Quality Through Coordination. In *Proceedings of the 2008 ACM Conference on Computer Supported Cooperative Work*, CSCW '08, pages 37–46, New York, NY, USA, 2008. ACM.

[91] Aniket Kittur, Boris Smus, Susheel Khamkar, and Robert E. Kraut. CrowdForge: Crowdsourcing Complex Work. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 43–52, New York, NY, USA, 2011. ACM.

[92] Kyungmin Lee, Jason Flinn, T.J. Giuli, Brian Noble, and Christopher Peplin. AMC: Verifying User Interface Properties for Vehicular Applications. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '13, pages 1–12, New York, NY, USA, 2013. ACM.

[93] Jialiu Lin, Shahriyar Amini, Jason I. Hong, Norman Sadeh, Janne Lindqvist, and Joy Zhang. Expectation and Purpose: Understanding Users' Mental Models of Mobile App Privacy through Crowdsourcing. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp '12, pages 501–510, New York, NY, USA, 2012. ACM.

[94] Greg Little, Lydia B. Chilton, Max Goldman, and Robert C. Miller. Exploring Iterative and Parallel Human Computation Processes. In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, HCOMP '10, pages 68–76, New York, NY, USA, 2010. ACM.

[95] Hiroshi Lockheimer. Android and Security. `http://googlemobile.blogspot.com/2012/02/android-and-security.html`.

[96] Michael R. Lyu. *Software Fault Tolerance*. Wiley, 1995.

[97] Tim Mackenzie. App store fees, percentages, and payouts: What developers need to know. *Tech Republic*, May 2012. `http://www.techrepublic.com/blog/software-engineer/app-store-fees-percentages-and-payouts-what-developers-need-to-know/1205/`.

[98] Jason Mick. Android has 97 Percent of Mobile Malware, But Nearly None in the U.S. *Daily Tech*, 2014. `http://www.dailytech.com/Android+has+97+Percent+of+Mobile+Malware+But+Nearly+None+in+the+US/article34595.htm`.

[99] David L. Mills. Improved Algorithms for Synchronizing Computer Network Clocks. *IEEE/ACM Trans. Netw.*, 3(3):245–254, June 1995.

[100] Andreas Moser, Christopher Kruegel, and Engin Kirda. Limits of Static Analysis for Malware Detection. In *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, pages 421 –430, dec. 2007.

[101] National Security Agency. Mobility Capability Package. February 2012.

[102] Mohammad Nauman, Sohail Khan, and Xinwen Zhang. Apex: Extending Android Permission Model and Enforcement with User-defined Runtime Constraints. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '10, pages 328–332, New York, NY, USA, 2010. ACM.

[103] Saul B. Needleman and Christian D. Wunsch. A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins. *Journal of Molecular Biology*, 48(3):443 – 453, 1970.

[104] Phil Nickinson. Google unveils 'Editor's Choice' badges for top, trustworthy apps. May 2011. `http://www.androidcentral.com/google-unveils-editors-choice-badges-top-trustworthy-apps`.

[105] Jon Oberheide and Charlie Miller. Dissecting the Android Bouncer. `https://jon.oberheide.org/files/summercon12-bouncer.pdf`.

[106] Machigar Ongtang, Stephen McLaughlin, William Enck, and Patrick McDaniel. Semantically Rich Application-Centric Security in Android. In *Computer Security Applications Conference, 2009. ACSAC '09. Annual*, pages 340–349, Dec 2009.

[107] Courteney Palis. Apple App Store Now Features 'Editors' Choice' And 'App Of The Week'. *The Huffington Post*, May 2012. http://www.huffingtonpost.com/2012/05/25/apple-app-store-new-features_n_1545972.html.

[108] Rahul Pandita, Xusheng Xiao, Wei Yang, William Enck, and Tao Xie. WHYPER: Towards Automating Risk Assessment of Mobile Applications. In *Proceedings of the 22Nd USENIX Conference on Security*, SEC'13, pages 527–542, Berkeley, CA, USA, 2013. USENIX Association.

[109] Kevin Parrish. Storm8: We're Not Collecting Private iPhone Data. Dec 2009. http://www.tomsguide.com/us/iPhone-Developer-Storm8-Private-Data,news-5283.html.

[110] Paul Pearce, Adrienne Porter Felt, Gabriel Nunez, and David Wagner. AdDroid: Privilege Separation for Applications and Advertisers in Android. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '12, pages 71–72, New York, NY, USA, 2012. ACM.

[111] Georgios Portokalidis, Philip Homburg, Kostas Anagnostakis, and Herbert Bos. Paranoid Android: Versatile Protection for Smartphones. In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, pages 347–356, New York, NY, USA, 2010. ACM.

[112] Alexander J. Quinn and Benjamin B. Bederson. Human Computation: A Survey and Taxonomy of a Growing Field. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 1403–1412, New York, NY, USA, 2011. ACM.

[113] Parameswaran Ramanathan, Kang G. Shin, and Ricky W. Butler. Fault-Tolerant Clock Synchronization in Distributed Systems. *Computer*, 23(10):33–42, Oct 1990.

[114] Vaibhav Rastogi, Yan Chen, and William Enck. AppsPlayground: Automatic Security Analysis of Smartphone Applications. In *Proceedings of the Third ACM Conference on Data and Application Security and Privacy*, CODASPY '13, pages 209–220, New York, NY, USA, 2013. ACM.

[115] Sanae Rosen, Zhiyun Qian, and Z. Morely Mao. AppProfiler: A Flexible Method of Exposing Privacy-related Behavior in Android Applications to End Users. In *Proceedings of the Third ACM Conference on Data and Application Security and Privacy*, CODASPY '13, pages 221–232, New York, NY, USA, 2013. ACM.

[116] Dan Rowinski. Apple iOS App Store Adding 20,000 Apps A Month, Hits 40 Billion Downloads. Feb 2014. http://readwrite.com/2013/01/07/apple-app-store-growing-by.

[117] Jeffrey M. Rzeszotarski and Aniket Kittur. Instrumenting the Crowd: Using Implicit Behavioral Measures to Predict Task Performance. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 13–22, New York, NY, USA, 2011. ACM.

[118] Golam Sarwar, Olivier Mehani, Roksana Boreli, and Mohamed˜Ali Kaafar. On the Effectiveness of Dynamic Taint Analysis for Protecting Against Private Information Leaks on Android-based Devices. In Pierangela Samarati, editor, *Proc. SECRYPT 2013*, pages 461–467. ACM SIGSAC, SciTePress, July 2013.

[119] Asaf Shabtai, Uri Kanonov, Yuval Elovici, Chanan Glezer, and Yael Weiss. "Andromaly": A Behavioral Malware Detection Framework for Android Devices. *J. Intell. Inf. Syst.*, 38(1):161–190, February 2012.

[120] Tyler Shields. Mobile Apps Invading Your Privacy. *Vercode Blog*, Apr 2011. http://www.veracode.com/blog/2011/04/mobile-apps-invading-your-privacy/.

[121] Eric Smith. iPhone Applications & Privacy Issues: An Analysis of Application Transmission of iPhone Unique Device Identifiers (UDIDs). Oct 2010. http://www.pskl.us.

[122] Tommi Takala, Mika Katara, and Julian Harty. Experiences of System-Level Model-Based GUI Testing of an Android Application. In *Proceedings of the 2011 Fourth IEEE International Conference on Software Testing, Verification and Validation*, ICST '11, pages 377–386, Washington, DC, USA, 2011. IEEE Computer Society.

[123] Jennifer Tam, Robert W. Reeder, and Stuart Schechter. I'm Allowing What? Technical Report MSR-TR-2010-54, Microsoft Research, 2010.

[124] Arun Thampi. Path uploads your entire iPhone address book to its servers. Feb 2012. http://mclov.in/2012/02/08/path-uploads-your-entire-address-book-to-their-servers.html.

[125] Scott Thurm and Yukari Iwatani Kane. The Journal's Cellphone Testing Methodology. *The Wall Street Journal*, Dec 2010. http://online.wsj.com/news/articles/SB10001424052748704034804576025951767626460.

[126] Scott Thurm and Yukari Iwatani Kane. Your Apps Are Watching You. *The Wall Street Journal*, Dec 2010.

[127] United States of America. Consent Decree And Order For Civil Penalties, Permanent Injunction And Other Relief. Court Case 13-cv-00448-RS, United States District Court Northern District of California San Francisco Division, February 2013.

[128] Jeff Vance. Free mobile apps put your BYOD strategies at risk. *Computerworld*, September 2012.

[129] Timothy Vidas, Nicolas Christin, and Lorrie Faith Cranor. Curbing Android Permission Creep. In *In W2SP*, 2011.

[130] Luis von Ahn, Manuel Blum, and John Langford. Telling Humans and Computers Apart Automatically. *Commun. ACM*, 47(2):56–60, February 2004.

[131] Xuetao Wei, Lorenzo Gomez, Iulian Neamtiu, and Michalis Faloutsos. ProfileDroid: Multi-layer Profiling of Android Applications. In *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking*, Mobicom '12, pages 137–148, New York, NY, USA, 2012. ACM.

[132] WMUR. Police: Thieves Robbed Homes Based On Facebook, Social Media Sites. Sep 2010. http://www.wmur.com/r/24943582/detail.html.

[133] Tingxin Yan, Vikas Kumar, and Deepak Ganesan. CrowdSearch: Exploiting Crowds for Accurate Real-time Image Search on Mobile Phones. In *Proceedings of the 8th International Conference*

on Mobile Systems, Applications, and Services, MobiSys '10, pages 77–90, New York, NY, USA, 2010. ACM.

[134] Liu Yang, Nader Boushehrinejadmoradi, Pallab Roy, Vinod Ganapathy, and Liviu Iftode. Short Paper: Enhancing Users' Comprehension of Android Permissions. In *Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '12, pages 21–26, New York, NY, USA, 2012. ACM.

[135] Jeffrey Zeldman. *Taking Your Talent to the Web: A Guide for the Transitioning Designer*. New Riders Publishing, 2001.

[136] Cong Zheng, Shixiong Zhu, Shuaifu Dai, Guofei Gu, Xiaorui Gong, Xinhui Han, and Wei Zou. SmartDroid: An Automatic System for Revealing UI-based Trigger Conditions in Android Applications. In *Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '12, pages 93–104, New York, NY, USA, 2012. ACM.

[137] Yajin Zhou, Xinwen Zhang, Xuxian Jiang, and Vincent W. Freeh. Taming Information-Stealing Smartphone Applications (on Android). In *Proceedings of the 4th International Conference on Trust and Trustworthy Computing*, TRUST'11, pages 93–107, Berlin, Heidelberg, 2011. Springer-Verlag.

A Proposal to Crowdsource End-User Expectations

Appendix A brings forth the first proposal to ever present the use of crowdsourcing to harvest user expectations regarding the privacy implications associated with using mobile apps. This proposal was written by Amini in February of 2011 for application to the Qualcomm Innovation Fellowship.

Crowdsourcing Mobile Application Instrumentation

Shahriyar Amini
Electrical & Computer Engineering
Carnegie Mellon University
samini@ece.cmu.edu

Jialiu Lin
Computer Science Department
Carnegie Mellon University
jialiul@cs.cmu.edu

1 Overview

With the widespread use of smart phones, mobile phone applications have gained mainstream popularity. In 2010 alone, there have been billions of application downloads from mobile application stores such as the Apple App Store and the Android Market, generating billions of dollars in revenue. These statistics are forecasted to more than double in 2011 [5]. The rapid increase in the use and development of mobile applications is a clear indicator of the widespread adoption of mobile application. Unfortunately, the use of mobile applications is coupled with increased risk to personal privacy through the unauthorized use of private information.

Mobile applications have the advantage of knowing about the user's context when providing content and services. It is not uncommon for mobile devices to feature a multi-axis accelerometer, compass, proximity sensor, light sensor, gyroscope, and GPS. Through these sensors, mobile applications have access to a wide variety of features regarding the user's context. More specifically, it is becoming increasingly easier for mobile applications to inquire and infer the user's location and context through the mobile platform's APIs. The user's context has been used effectively through various innovative methods to provide cutting-edge applications [1]. Nevertheless, the user's context and private information have also been misused in the past towards economical gain, physical stalking [2], and home invasions [10].

With great potential for providing users with application and services, and also for malicious conduct, policing the use of the mobile device's resources and the user's context is a high priority concern. Apple, the owner of the largest mobile application market, requires applications to pass through a vetting process before making them available. However, the nature of the process is not disclosed to the public. In fact, there have been malicious applications that had previously passed through the process, before being removed from the App Store in response to user complaints [9]. On the other hand, the Google owned Android Market, relies on the developers to specify the resources used in an application manifest and directly reports this information to the end user. However, it is questionable whether users have the knowhow to clearly comprehend how such resources can be used in malicious ways.

To address this problem, we propose a new solution to better understand application resource requirements, functionality, and private information leakage. Namely, through crowdsourcing application usage, we plan to automate the instrumentation of a large range of mobile applications and learn about the application behavior as well as the user's knowledge of the application conduct. We aim to devise a large-scale instrumentation platform that collects such information through the use of multiple isolated mobile platform emulators working in parallel. The data collected will be used to better inform user's decisions with regards to downloading and installing applications, to enable improved application recommendation systems, and to enhance the design of interfaces that inform users of the application behavior and resource usage.

2 State of the Art

Through the use of instrumentation platforms, two separate research efforts have looked at application resource usage and leakage of private user information [6, 8]. Smith and Thurm both investigate popular mobile applications by setting up testbenches that capture and analyze network packet transmissions to detect the transfer of private information from the mobile application to external servers. It is not surprising that both studies found numerous instances of applications that share the user's private information

to third parties beyond the user and the service provider endpoint. Although our approach will be similar in that we aim to instrument mobile applications, we propose an automated large-scale solution that allows the instrumentation of ten to hundreds of applications in parallel. Such a solution would allow the instrumentation of more than just the popular applications.

Recently, researchers have applied information flow tracking approaches to analyze application behavior. [3] uses a static analysis technique called PiOS on Apple's iOS applications. In [3], Egele reports that while most applications do not secretly leak sensitive information, majority of applications leak the device ID, which can provide detailed information about a user's habits. [4] uses a dynamic taint analysis technique to instrument Android applications, proposing TaintDroid. Enck's study reveals that two-thirds of applications in the study exhibit suspicious behavior, and half of the applications report the users' locations to remote advertising servers. While information flow tracking approaches are useful and effective, they do not scale well on their own. We see such approaches benefiting from the crowdsourcing instrumentation technique that we propose here to enable the data collection and analysis of thousands of applications.

Finally, there has been work exploring effective means of informing users regarding application resource usage through interface design [7]. Tam finds that users prefer designs that present resources visually, e.g. icons or pictures, allowing a quick search by the user. The data collected and analyzed through our automated crowdsourcing instrumentation would guide the design of interfaces through identifying what information is most critical when it comes to applications, resource usage, and sensitive user information.

3 Research Thrusts

It is our goal to automate and crowdsource the instrumentation of mobile applications. To this end, we plan to take advantage of Amazon's Mechanical Turk (MTurk) service. MTurk allows requester clients to define human intelligence tasks (HIT) that are then completed by workers who have signed up through the service. Clearly, some level of training and qualification is necessary for worker clients to interact with mobile applications. MTurk allows for the workers qualifications to be gauged through qualification tasks, which are also defined by the requester clients.

Considering that a pool of study participants is available to us through MTurk, a number of challenges remain in creating useful tasks that we can instrument and capture. First, there are a variety of mobile applications with different features, interfaces, and resource requirements. Clearly there is no single task that could cover a wide range of applications. Second, it is necessary to isolate various application instances so that multiple instance of various applications can be run simultaneously to allow for large scale instrumentation. Third, it is not clear what is normal behavior for an application. This challenge presents itself especially in the case of TaintDroid and PiOS. Instrumentation of an application and information flow analysis do not necessarily define what is normal behavior for the application. For instance, sharing the users location with others may seem to be abnormal, however, it is the business model behind widely used services such as foursquare and Facebook Places.

We will automate and crowdsource instrumentation while addressing the aforementioned challenges using a solution comprised of MTurk, isolated virtual machines, isolated mobile platform emulators per application, and user generated feedback loops. The end-to-end instrumentation is a three phase process. First, we define a HIT where we request participants to define the features of a mobile application and tasks that can be performed using the mobile application. In the second phase, we provide participants with an isolated emulator that provides one of the applications to be instrumented. The participant uses the emulator to perform the tasks defined in the first phase. While this happens, the user interaction and network transmissions are recorded for further analysis. Each emulator runs on its own isolated virtual machine. At this point, we run various analysis algorithms which infer application behavior and resource usage based on the captured network transmissions and user interactions. Finally, in the third phase, we present the inference produced in the second phase to participants as yet another HIT. In this phase, participants vote on the inference made by our algorithms regarding the application behavior.

Participants also vote on whether the application behavior is normal or abnormal based on the inference provided.

The three phase instrumentation process above allows for automated crowdsourced instrumentation. Specifically, the features and capabilities of the applications are defined by MTurk participants. Applications are isolated through the use of isolated emulators per application and virtual machines. Finally, abnormal behavior and the quality of inference is judged by MTurk participants. The second phase involves the bulk of the instrumentation. Specifically, we will rely on network packet capture, text alignment, and language technology techniques to detect the transmission of critical user and device information. We also plan to incorporate TaintDroid as part of our instrumentation as the TaintDroid source code has been made available to the public.

4 Researcher Qualifications

We acknowledge that the development and implementation of the end-to-end instrumentation platform described in Section 3 will be challenging. However, both researchers who will be involved in this project have acquired immense experience building numerous end-to-end systems in both academic and industry research lab environments. Shahriyar "Shah" Amini has a strong systems background, having built systems covering mobile location-based search, application usage and location monitoring, Caché: a mobile privacy-enhancing solution, and a hardware-based information flow taint analyzer. Although this research effort is in its nascent, Shah has produced a preliminary isolated virtual machine and emulator setup for the Android platform, capable of network transmission capture.

Jialiu Lin has a strong research background in context-aware mobile computing. Her prior research projects cover a broad spectrum from system issues, such as mobile sensing and energy efficiency, to human factors, such as users privacy concerns and preferences. These previous experiences have prepared her well with skills in system building, data analysis (machine learning and data mining), and designing and conducting user studies. Apart from her system building experience, Jialiu's strong background in machine learning and data mining will enable a rich set of instrumentation techniques based on network packet and user interaction inference.

5 Expected Outcomes

We envision this research project to span over several years. However, we see a fully functional automated crowdsourcing instrumentation platform within the one year horizon. We expect to collect and analyze several months of application instrumentation and user interaction data within the first year. To our knowledge, our approach will be the first of its kind. Our findings, which would be informed by a large-scale collection of mobile application instrumentation data, will inform mobile application market providers and developers regarding the critical aspects of applications with regards to resource usage and sensitive information transmission. In addition, our research results will enable the design of improved user interfaces, capable of better informing users about the applications that they purchase and install.

References

[1] Erin Biba. Inside the GPS Revolution: 10 Applications That Make the Most of Location. *Wired Magazine*, 17(2), 2009.

[2] Dailynews. How Cell Phone Helped Cops Nail Key Murder Suspect Secret Pings That Gave Bouncer Away. *Dailynews*, Mar 2006.

[3] Manuel Egele, Christopher Kruegel, Engin Kirda, and Giovanni Vigna. PiOS : Detecting privacy leaks in iOS applications. In *NDSS '11*, 2011.

[4] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anomol N. Sheth. TaintDroid: An Information-Flow Tracking System for Realtime PrivacyMonitoring on Smartphones. In *OSDI '10*, 2010.

[5] Christy Pettey. Gartner Says Worldwide Mobile Application Store Revenue Forecast to Surpass $15 Billion in 2011. http://www.gartner.com/it/page.jsp?id=1529214, Jan 2011.

[6] Eric Smith. iPhone Applications & Privacy Issues: An Analysis of Application Transmission of iPhone Unique Device Identifiers (UDIDs). http://www.pskl.us, Oct 2010.

[7] Jennifer Tam, Robert W. Reeder, and Stuart Schechter. I'm Allowing What? Technical Report MSR-TR-2010-54, Microsoft Research, 2010.

[8] Scott Thurm and Yukari Iwatani Kane. Your Apps Are Watching You. *The Wall Street Journal*, Dec 2010.

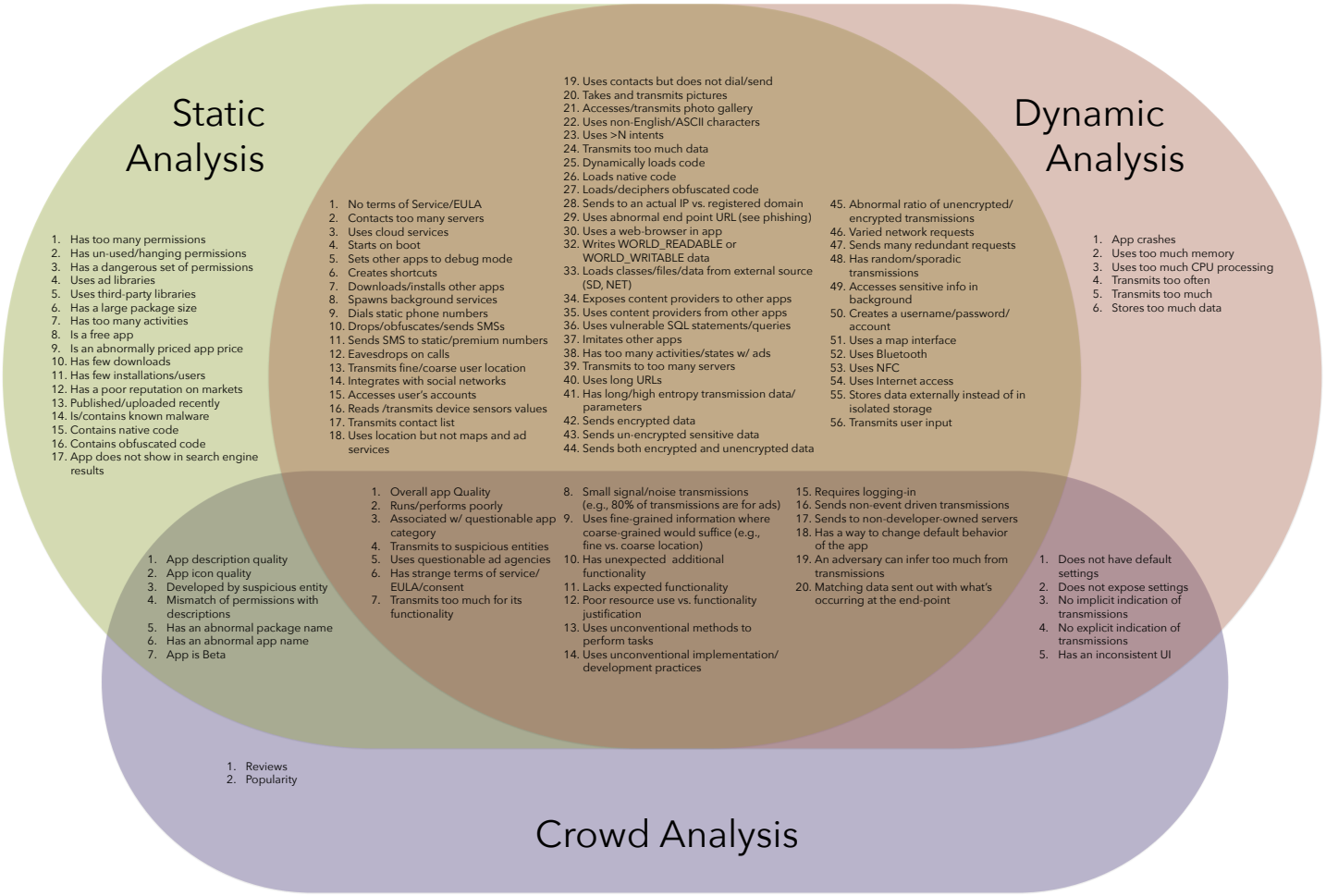
[9] Wired. Apple Approves, Pulls Flashlight App with Hidden Tethering Mode. *Wired*, Jul 2010.

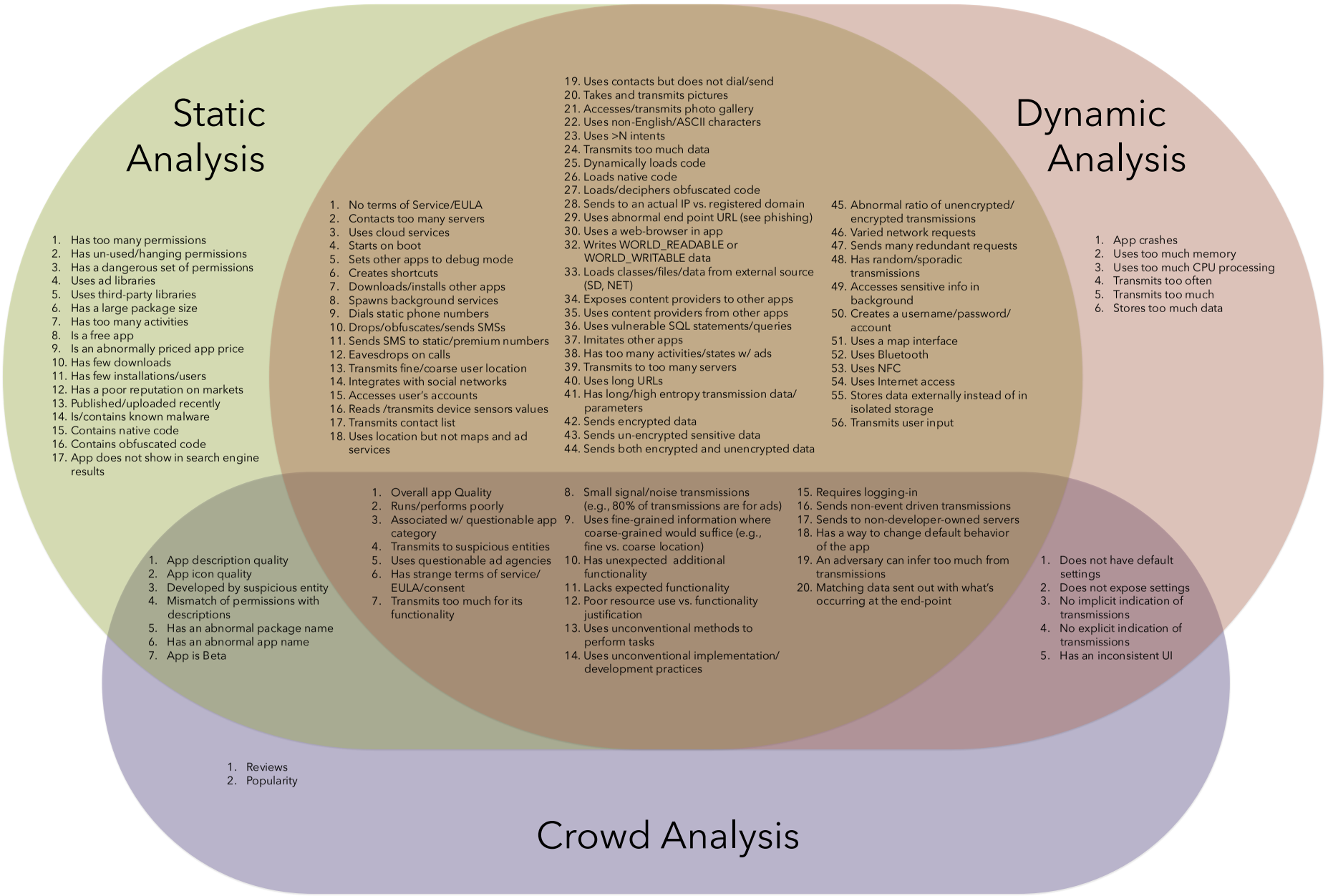
[10] WMUR. Police: Thieves Robbed Homes Based On Facebook, Social Media Sites. http://www.wmur.com/r/24943582/detail.html, Sep 2010.

B Heuristic Categorization Based on Analysis Technique

This appendix shows a summary of the heuristics compiled by the author based on prior work, a manual inspection of over 40 popular apps, and interviews with experts. The heuristics are categorized into a Venn diagram based on the type of analysis technique that could be performed on an app to determine the result of the heuristic. As some heuristics may need a combination of techniques, e.g., static and dynamic analysis, static or crowd analysis, heuristics belonging to more than one category could use any combination of the techniques. While the categorization should extend to other platforms, it was performed for Android.

The Venn diagram is provided in horizontal and vertical format for the reader's viewing/printing convenience. The horizontal and the vertical formats both present the same content.


Static Analysis
1. Has too many permissions
2. Has un-used/hanging permissions
3. Has a dangerous set of permissions
4. Uses ad libraries
5. Uses third-party libraries
6. Has a large package size
7. Has too many activities
8. Is a free app
9. Is an abnormally priced app price
10. Has few downloads
11. Has few installations/users
12. Has a poor reputation on markets
13. Published/uploaded recently
14. Is/contains known malware
15. Contains native code
16. Contains obfuscated code
17. App does not show in search engine results
1. No terms of Service/EULA
2. Contacts too many servers
3. Uses cloud services
4. Starts on boot
5. Sets other apps to debug mode
6. Creates shortcuts
7. Downloads/installs other apps
8. Spawns background services
9. Dials static phone numbers
10. Drops/obfuscates/sends SMSs
11. Sends SMS to static/premium numbers
12. Eavesdrops on calls
13. Transmits fine/coarse user location
14. Integrates with social networks
15. Accesses user's accounts
16. Reads /transmits device sensors values
17. Transmits contact list
18. Uses location but not maps and ad services
19. Uses contacts but does not dial/send
20. Takes and transmits pictures
21. Accesses/transmits photo gallery
22. Uses non-English/ASCII characters
23. Uses >N intents
24. Transmits too much data
25. Dynamically loads code
26. Loads native code
27. Loads/deciphers obfuscated code
28. Sends to an actual IP vs. registered domain
29. Uses abnormal end point URL (see phishing)
30. Uses a web-browser in app
32. Writes WORLD_READABLE or WORLD_WRITABLE data
33. Loads classes/files/data from external source (SD, NET)
34. Exposes content providers to other apps
35. Uses content providers from other apps
36. Uses vulnerable SQL statements/queries
37. Imitates other apps
38. Has too many activities/states w/ ads
39. Transmits to too many servers
40. Uses long URLs
41. Has long/high entropy transmission data/ parameters
42. Sends encrypted data
43. Sends un-encrypted sensitive data
44. Sends both encrypted and unencrypted data
45. Abnormal ratio of unencrypted/ encrypted transmissions
46. Varied network requests
47. Sends many redundant requests
48. Has random/sporadic transmissions
49. Accesses sensitive info in background
50. Creates a username/password/ account
51. Uses a map interface
52. Uses Bluetooth
53. Uses NFC
54. Uses Internet access
55. Stores data externally instead of in isolated storage
56. Transmits user input
Dynamic Analysis
1. App crashes
2. Uses too much memory
3. Uses too much CPU processing
4. Transmits too often
5. Transmits too much
6. Stores too much data
1. App description quality
2. App icon quality
3. Developed by suspicious entity
4. Mismatch of permissions with descriptions
5. Has an abnormal package name
6. Has an abnormal app name
7. App is Beta
1. Overall app Quality
2. Runs/performs poorly
3. Associated w/ questionable app category
4. Transmits to suspicious entities
5. Uses questionable ad agencies
6. Has strange terms of service/ EULA/consent
7. Transmits too much for its functionality
8. Small signal/noise transmissions (e.g., 80% of transmissions are for ads)
9. Uses fine-grained information where coarse-grained would suffice (e.g., fine vs. coarse location)
10. Has unexpected additional functionality
11. Lacks expected functionality
12. Poor resource use vs. functionality justification
13. Uses unconventional methods to perform tasks
14. Uses unconventional implementation/ development practices
15. Requires logging-in
16. Sends non-event driven transmissions
17. Sends to non-developer-owned servers
18. Has a way to change default behavior of the app
19. An adversary can infer too much from transmissions
20. Matching data sent out with what's occurring at the end-point
1. Does not have default settings
2. Does not expose settings
3. No implicit indication of transmissions
4. No explicit indication of transmissions
5. Has an inconsistent UI
1. Reviews
2. Popularity
Crowd Analysis


Static Analysis
1. Has too many permissions
2. Has un-used/hanging permissions
3. Has a dangerous set of permissions
4. Uses ad libraries
5. Uses third-party libraries
6. Has a large package size
7. Has too many activities
8. Is a free app
9. Is an abnormally priced app price
10. Has few downloads
11. Has few installations/users
12. Has a poor reputation on markets
13. Published/uploaded recently
14. Is/contains known malware
15. Contains native code
16. Contains obfuscated code
17. App does not show in search engine results
Dynamic Analysis
1. App crashes
2. Uses too much memory
3. Uses too much CPU processing
4. Transmits too often
5. Transmits too much
6. Stores too much data
1. No terms of Service/EULA
2. Contacts too many servers
3. Uses cloud services
4. Starts on boot
5. Sets other apps to debug mode
6. Creates shortcuts
7. Downloads/installs other apps
8. Spawns background services
9. Dials static phone numbers
10. Drops/obfuscates/sends SMSs
11. Sends SMS to static/premium numbers
12. Eavesdrops on calls
13. Transmits fine/coarse user location
14. Integrates with social networks
15. Accesses user's accounts
16. Reads /transmits device sensors values
17. Transmits contact list
18. Uses location but not maps and ad services
19. Uses contacts but does not dial/send
20. Takes and transmits pictures
21. Accesses/transmits photo gallery
22. Uses non-English/ASCII characters
23. Uses >N intents
24. Transmits too much data
25. Dynamically loads code
26. Loads native code
27. Loads/deciphers obfuscated code
28. Sends to an actual IP vs. registered domain
29. Uses abnormal end point URL (see phishing)
30. Uses a web-browser in app
32. Writes WORLD_READABLE or WORLD_WRITABLE data
33. Loads classes/files/data from external source (SD, NET)
34. Exposes content providers to other apps
35. Uses content providers from other apps
36. Uses vulnerable SQL statements/queries
37. Imitates other apps
38. Has too many activities/states w/ ads
39. Transmits to too many servers
40. Uses long URLs
41. Has long/high entropy transmission data/ parameters
42. Sends encrypted data
43. Sends un-encrypted sensitive data
44. Sends both encrypted and unencrypted data
45. Abnormal ratio of unencrypted/ encrypted transmissions
46. Varied network requests
47. Sends many redundant requests
48. Has random/sporadic transmissions
49. Accesses sensitive info in background
50. Creates a username/password/ account
51. Uses a map interface
52. Uses Bluetooth
53. Uses NFC
54. Uses Internet access
55. Stores data externally instead of in isolated storage
56. Transmits user input
1. Overall app Quality
2. Runs/performs poorly
3. Associated w/ questionable app category
4. Transmits to suspicious entities
5. Uses questionable ad agencies
6. Has strange terms of service/ EULA/consent
7. Transmits too much for its functionality
8. Small signal/noise transmissions (e.g., 80% of transmissions are for ads)
9. Uses fine-grained information where coarse-grained would suffice (e.g., fine vs. coarse location)
10. Has unexpected additional functionality
11. Lacks expected functionality
12. Poor resource use vs. functionality justification
13. Uses unconventional methods to perform tasks
14. Uses unconventional implementation/ development practices
15. Requires logging-in
16. Sends non-event driven transmissions
17. Sends to non-developer-owned servers
18. Has a way to change default behavior of the app
19. An adversary can infer too much from transmissions
20. Matching data sent out with what's occurring at the end-point
1. App description quality
2. App icon quality
3. Developed by suspicious entity
4. Mismatch of permissions with descriptions
5. Has an abnormal package name
6. Has an abnormal app name
7. App is Beta
1. Does not have default settings
2. Does not expose settings
3. No implicit indication of transmissions
4. No explicit indication of transmissions
5. Has an inconsistent UI
1. Reviews
2. Popularity
Crowd Analysis

C Gort Task Extraction Human Intelligence Task Sample

This appendix presents the Mechanical Turk Human Intelligence Task (HIT) created to extract tasks from the Horoscope Android mobile application. The HIT includes questions asking for the crowd worker's Android device version, whether the worker has used the app before, the category of the app, and also a label for the sequence of screenshots presented. The first three questions can be used to verify a crowd worker's diligence in answering the HIT as the answers are simple to validate.

This task requires that you own and use an Android smartphone. You should have experience installing and running applications on your smartphone.

1. What version of Android do you run on your device? To find the version of Android on your device, go to **Settings** on your phone. Select **About phone**. The Android version is displayed as one of the items on the screen. Note the version number is a number separated by dots and does not include words or any other characters. If you use more than one Android smartphone, please enter the version for the device you use most often. (required)

Please read the application description carefully, look at the application screenshots, and answer the questions below.

App Name: Horoscope

Horoscope : The official Horoscope from horoscope.fr now available on your Android phone ! 100% FREE, 100% PRO !

#1 Top Horoscope !
#10+ Millions downloads !
Compatible with tablets !

Check your complete horoscopes for today, tomorrow and keep an eye on your horoscope of the week, your monthly horoscope and your yearly horoscope written by our best, most serious and experienced astrologers.

The horoscope texts are updated every day in real time, and you can consult them at any time without any limitations!

What can you expect from the stars ? For Love, Work, or Wellness ? Which key-dates are the essential for success this month ? Horoscope on Android is the horoscope application you must have... Download it now!

For any questions/features on the Horoscope app :
****
mobile-support@telemaque.fr
****

Explanation of requested permissions by Horoscope

- Approximate geographical location :
Your approximate geographical location is requested in order to help determine your astrological profile (automatically detects the places nearby).

- Network communication (show your WiFi and network status) :
Allow you to enjoy the best connection available (WiFi and/or Network) for the best possible browsing experience.

- Phone calls :
Allows you to keep the application open when a phone call or a text message interrupts your reading.

- Phone/SD CARD storage :
The information entered in the application will be kept on your Smartphone/SD card in order to give you a personalized content every time you open the application.

- E-mail accounts :
Allows you to share your horoscope easily by automatically filling the content of the e-mail and the e-mail address of the sender.

- Network communications (general internet access) :
Allows you to enjoy an up-to-date content (daily, weekly and monthly horoscopes).

- System tools :
Allows the application to run faster when your smartphone is running out of virtual memory.

2. Have you used this app before? (required)

- ◯ Yes
- ◯ No

3. What category do you think this mobile app should belong to? (required)

- ◯ Game application
- ◯ Non-game application
- ◯ Book, music, or video

Suppose you have installed and are using Horoscope on your Android smartphone. You just went through the following two screens on your device, starting with the screen on the left.

4. What task were you trying to accomplish **over the course of the above two screens**? (examples: search for friends, share a link, take notes, play a game) Note: If the two screens are too similar, please respond with respect to one screen. (required)

5. Please provide any other comments you may. We appreciate your input!

You must ACCEPT the HIT before you can submit the results.

D Gort Task Verification Human Intelligence Task Sample

This appendix presents the Mechanical Turk Human Intelligence Task (HIT) created to verify task labels for a screenshot sequence from the Horoscope Android mobile application. Similar to the task extraction sample, this HIT also presents questions to determine a crowd worker's dedication to answering the HIT.

This task requires that you own and use an Android smartphone. You should have experience installing and running applications on your smartphone.

1. What version of Android do you run on your device? To find the version of Android on your device, go to **Settings** on your phone. Select **About phone**. The Android version is displayed as one of the items on the screen. Note the version number is a number separated by dots and does not include words or any other characters. If you use more than one Android smartphone, please enter the version for the device you use most often. (required)

Please read the application description carefully, look at the application screenshots, and answer the questions below.

App Name: Horoscope

Horoscope : The official Horoscope from horoscope.fr now available on your Android phone ! 100% FREE, 100% PRO !

#1 Top Horoscope !
#10+ Millions downloads !
Compatible with tablets !

Check your complete horoscopes for today, tomorrow and keep an eye on your horoscope of the week, your monthly horoscope and your yearly horoscope written by our best, most serious and experienced astrologers.

The horoscope texts are updated every day in real time, and you can consult them at any time without any limitations!

What can you expect from the stars ? For Love, Work, or Wellness ? Which key-dates are the essential for success this month ? Horoscope on Android is the horoscope application you must have... Download it now!

For any questions/features on the Horoscope app :
****
mobile-support@telemaque.fr
****

Explanation of requested permissions by Horoscope

- Approximate geographical location :
Your approximate geographical location is requested in order to help determine your astrological profile (automatically detects the places nearby).

- Network communication (show your WiFi and network status) :
Allow you to enjoy the best connection available (WiFi and/or Network) for the best possible browsing experience.

- Phone calls :
Allows you to keep the application open when a phone call or a text message interrupts your reading.

- Phone/SD CARD storage :
The information entered in the application will be kept on your Smartphone/SD card in order to give you a personalized content every time you open the application.

- E-mail accounts :
Allows you to share your horoscope easily by automatically filling the content of the e-mail and the e-mail address of the sender.

- Network communications (general internet access) :
Allows you to enjoy an up-to-date content (daily, weekly and monthly horoscopes).

- System tools :
Allows the application to run faster when your smartphone is running out of virtual memory.

2. Have you used this app before? (required)

- ◯ Yes
- ◯ No

3. What category do you think this mobile app should belong to? (required)

- ◯ Game application
- ◯ Non-game application
- ◯ Book, music, or video

Suppose you have installed and are using Horoscope on your Android smartphone. You just went through the following two screens on your device, starting with the screen on the left. Think about the task you were trying to accomplish over the course of these two screens.

4. Select how well each of the options below describes what you were trying to accomplish **over the course of the above two screens**. Please take into account grammar and spelling. Note: If the two screens are too similar, please respond with respect to a single screen. (required)

	Extremely Poor	Below Average	Slightly Below Average	Average	Slightly Above Average	Above Average	Excellent
check my horoscope	○	○	○	○	○	○	○
checking out your life in the future	○	○	○	○	○	○	○
Find out astrological predictions for the Aries sign for today.	○	○	○	○	○	○	○
See what Aries says	○	○	○	○	○	○	○
selecting your sign	○	○	○	○	○	○	○
Get more info about aries.	○	○	○	○	○	○	○
I can click on my specific horoscope	○	○	○	○	○	○	○
Read my horoscope	○	○	○	○	○	○	○
Daily reading	○	○	○	○	○	○	○
Select "Above Average" here	○	○	○	○	○	○	○

5. Please provide any other comments you have. We appreciate your input!

You must ACCEPT the HIT before you can submit the results.

E Gort Resource Justification Human Intelligence Task Sample

This appendix presents the Mechanical Turk Human Intelligence Task (HIT) created to determine whether the use of sensitive resources by an app is legitimate. Specifically, this HIT asks crowd workers whether it makes sense for the the Horoscope app to access the user's phone number, exact location, approximate location, and unique device identifier while the user is checking her horoscope.

This task requires that you own and use an Android smartphone. You should have experience installing and running applications on your smartphone.

1. What version of Android do you run on your device? To find the version of Android on your device, go to **Settings** on your phone. Select **About phone**. The Android version is displayed as one of the items on the screen. Note the version number is a number separated by dots and does not include words or any other characters. If you use more than one Android smartphone, please enter the version for the device you use most often. (required)

Please read the application description carefully, look at the application screenshots, and answer the questions below.

App Name: Horoscope

Horoscope : The official Horoscope from horoscope.fr now available on your Android phone ! 100% FREE, 100% PRO !

#1 Top Horoscope !
#10+ Millions downloads !
Compatible with tablets !

Check your complete horoscopes for today, tomorrow and keep an eye on your horoscope of the week, your monthly horoscope and your yearly horoscope written by our best, most serious and experienced astrologers.

The horoscope texts are updated every day in real time, and you can consult them at any time without any limitations!

What can you expect from the stars ? For Love, Work, or Wellness ? Which key-dates are the essential for success this month ? Horoscope on Android is the horoscope application you must have... Download it now!

For any questions/features on the Horoscope app :
****
mobile-support@telemaque.fr
****

Explanation of requested permissions by Horoscope

- Approximate geographical location :
Your approximate geographical location is requested in order to help determine your astrological profile (automatically detects the places nearby).

- Network communication (show your WiFi and network status) :
Allow you to enjoy the best connection available (WiFi and/or Network) for the best possible browsing experience.

- Phone calls :
Allows you to keep the application open when a phone call or a text message interrupts your reading.

- Phone/SD CARD storage :
The information entered in the application will be kept on your Smartphone/SD card in order to give you a personalized content every time you open the application.

- E-mail accounts :
Allows you to share your horoscope easily by automatically filling the content of the e-mail and the e-mail address of the sender.

- Network communications (general internet access) :
Allows you to enjoy an up-to-date content (daily, weekly and monthly horoscopes).

- System tools :
Allows the application to run faster when your smartphone is running out of virtual memory.

2. Have you used this app before? (required)

- Yes
- No

3. What category do you think this mobile app should belong to? (required)

- Game application
- Non-game application
- Book, music, or video

Suppose you have installed and are using Horoscope on your Android smartphone. You learn that Horoscope uses your **phone number, exact location, approximate location, and unique device identifier** while you conduct the following task with the app:

check my horoscope

4. Would you expect Horoscope to use your phone number, exact location, approximate location, and unique device identifier while conducting the above task? (required)

- Yes
- No
- I don't know what 'check my horoscope' means.

5. How comfortable do you feel with Horoscope using your phone number, exact location, approximate location, and unique device identifier for the task? (required)

- Very comfortable
- Somewhat comfortable
- Somewhat uncomfortable
- Very uncomfortable

6. Please provide any other comments you have. We appreciate your input!

You must ACCEPT the HIT before you can submit the results.

F Gort v2 User Study Protocol

Appendix F presents the protocol for the user study with Gort v2. Researchers should follow this same protocol if they are interested in replicating the study with Gort v2. Appendix G offers the questions presented to the participants in the form of a web-based survey as indicated in the protocol.

Italicized text is instructions to the investigator.

Introduce yourself. Engage in informal chit-chat and introduce who you are and why you are there.

- Thank you for being in the study.
- My name is _____, and I will be outlining the study for you and answering any questions that you may have.
- We're researching how students analyze mobile applications and investigate potentially sensitive application behavior.
- I have this script to read just to make sure I remember to do everything.
- If it is okay with you, I will be recording audio during the interview, just in case I forget to write down any important responses. This recording will not be shared with anyone else. I will be the only person who can listen to it.

Tell them about the interview.

- Today we will be pretending that you are an independent consultant for privacy and security. Your goal is to generate a report for a company whose employees are using applications that you will be inspecting. You will be looking at 3 Android applications and inspecting their behavior.
- At the end of the study, we'll immediately arrange for $30 compensation.
- We'll be done in about an hour and a half.

Consent Form
If that sounds good, please go ahead and sign this consent form.

Start the audio recorder if the user gave consent.

Point the participant to the survey.

Please fill out the following demographics page on this web-based survey.

Show the instruction video for how to use Gort.

Each analyst will get 3 applications: 1 test app to practice using the interface (WeatherBug), 1 app with sensitive but necessary transmissions (Yelp), 1 complex app with sensitive but unnecessary transmissions (Horoscope). The 2nd and 3rd app should be counter balanced. Participants can ask clarification questions regarding the practice task.

Imagine that you are an independent consultant for privacy and security. You are generating a report for a company whose employees are using the following apps.

You have 15 minutes to answer 26 questions regarding the app on the web-based survey. 22 of the questions are graded as indicated on the survey. Try to answer as many of these questions correctly as you can in the allotted amount of time.

Please read the questions and think aloud as you respond to them.

Repeat the above for all three apps.

Go to the final steps of the survey.

Here are some final questions regarding the general aspects of the tool. Please check back in once you have completed responding to them.

Following the end of the study and the survey, ask the questions below. Probe for more information based on the participant's responses.

What did you like about the tool?

What would you improve about the tool? Would you add or remove any features?

If you have used any other tools to analyze apps in the past, how does Gort compare to them?

Did you use any of the information on the 'Crowd' tab?

Can you come up with any reasons or scenarios why someone would use the 'Crowd' tab?

Any other feedback/comments?

G Gort v2 User Study App Analysis Questions

Appendix G presents the set of questions presented to participants during the user study with Gort v2. The questions were presented in the form of an online Survey Gizmo survey that participants followed along during the study. The task questions were asked for three different apps. The first app analysis task was a practice round for participants to become familiar with Gort. The timed questions were randomized for each run through the survey. The follow-up questions about Gort were asked at the end of the study, once the participant had analyzed all three applications.

Task Questions

17) What is the application name?*

_____

18) Have you heard of this app before?*
( ) Yes
( ) No

19) Have you used this app before?*
( ) Yes
( ) No

20) Briefly describe what is the main purpose of the app. (One sentence is enough)*

_____
_____
_____
_____

The next 22 questions are optional. Please try to correctly answer as many of them as you can in the allotted time.

21) Does the application access the Internet?
( ) Yes
( ) No
( ) I don't know.

22) Does the application send encrypted data?
( ) Yes
( ) No
( ) I don't know.

23) Does the application send un-encrypted data?
( ) Yes
( ) No
( ) I don't know.

24) How many permissions does the app use?
( ) 0
( ) 1-2
( ) 3-5
( ) 6-10
( ) 11+

25) How many servers does the app contact?

( ) 0
( ) 1-2
( ) 3-5
( ) 6-10
( ) 11+

26) What kind of sensitive information does the application use? (If the app uses more than three types of sensitive information, providing three types is sufficient)

_____
_____
_____
_____

27) Who develops the application?

_____

28) Who owns the servers that the app communicates with?

_____

29) Does the application transmit user's location?
( ) Yes
( ) No
( ) I don't know.

30) Does the application transmit the smartphone's device id?
( ) Yes
( ) No
( ) I don't know.

31) Does the application transmit the user's phone number?
( ) Yes
( ) No
( ) I don't know.

32) Does the app use any third-party libraries?
( ) Yes
( ) No
( ) I don't know.

33) Does the app contain native code (code that would run on the actual hardware rather than the JVM)?
( ) Yes
( ) No
( ) I don't know.

34) Can the application eavesdrop on calls?

( ) Yes
( ) No
( ) I don't know.

35) Can the application send or receive SMSs?
( ) Yes
( ) No
( ) I don't know.

36) Does the application use any cloud services (e.g., Amazon EC2, Google AppEngine, DropBox)?
( ) Yes
( ) No
( ) I don't know.

37) Does the application send redundant requests?
( ) Yes
( ) No
( ) I don't know.

38) Does the application send to a non-mapped IP (i.e., the IP address does not have a URL associated with it)?
( ) Yes
( ) No
( ) I don't know.

39) Does the app have a dangerous set of permissions? (Please elaborate)
_____
_____
_____
_____

40) Does the application use the smartphone's camera?
( ) Yes
( ) No
( ) I don't know.

41) Does the application run any services in the background?
( ) Yes
( ) No
( ) I don't know.

42) Does the application use a map interface?
( ) Yes
( ) No
( ) I don't know.

You have completed the optional questions section of this page. If you still have time, feel free to look over any questions you are not sure about. Otherwise, please check in with the study instructor.

43) What tasks can you do with this application?*

_____
_____
_____
_____

44) Do the permissions requested by the app seem justified based on the app description?*

( ) Yes
( ) No
( ) I don't know.

45) Imagine you have installed and use this app on your smartphone, what privacy concerns would you have about the app?*

_____
_____
_____
_____

46) What would other people think about the privacy implications of using this app?*

_____
_____
_____
_____

47) Why do you think the app transmits the information that it does?*

[ ] Functionality
[ ] Ads
[ ] Analytics (e.g., deciding how users use the app)
[ ] Social (e.g., sharing with friends, twitter)
[ ] Malicious Intent
[ ] I don't know.
[ ] Other: _____

48) Please select how well you agree with each of the following statements.*

	Strongly disagree	**Disagree**	**Neutral**	**Agree**	**Strongly agree**
It was easy to find out	( )	( )	( )	( )	( )

what sensitive information was sent.					
I am confident in my responses to the questions about the app.	( )	( )	( )	( )	( )
Considering the privacy implications of using this app, I would expect users to be comfortable with using this application.	( )	( )	( )	( )	( )

Questions about Gort

49) Please select how well you agree with each of the following statements.*

	Strongly disagree	**Disagree**	**Neutral**	**Agree**	**Strongly agree**
The tool offered enough details to answer questions about application privacy.	( )	( )	( )	( )	( )

The user interface (UI) was easy to use and follow.	( )	( )	( )	( )	( )
I would consider using this tool again to analyze application privacy.	( )	( )	( )	( )	( )
I found the 'Heuristics' useful.	( )	( )	( )	( )	( )
I found the 'Crowd' tab useful.	( )	( )	( )	( )	( )

50) What did you like about the tool?*

_____
_____
_____
_____

51) What would you improve about the tool? Would you add or remove any features? (Please elaborate)*

_____
_____
_____
_____

52) If you have used any other tools to analyze apps in the past, how does Gort compare to them?

_____
_____
_____
_____

53) Did you use any of the information on the 'Crowd' tab? (Please elaborate)

_____
_____
_____
_____

54) Can you come up with any reasons or scenarios why someone would use the 'Crowd' tab? (Please elaborate)

_____
_____
_____
_____

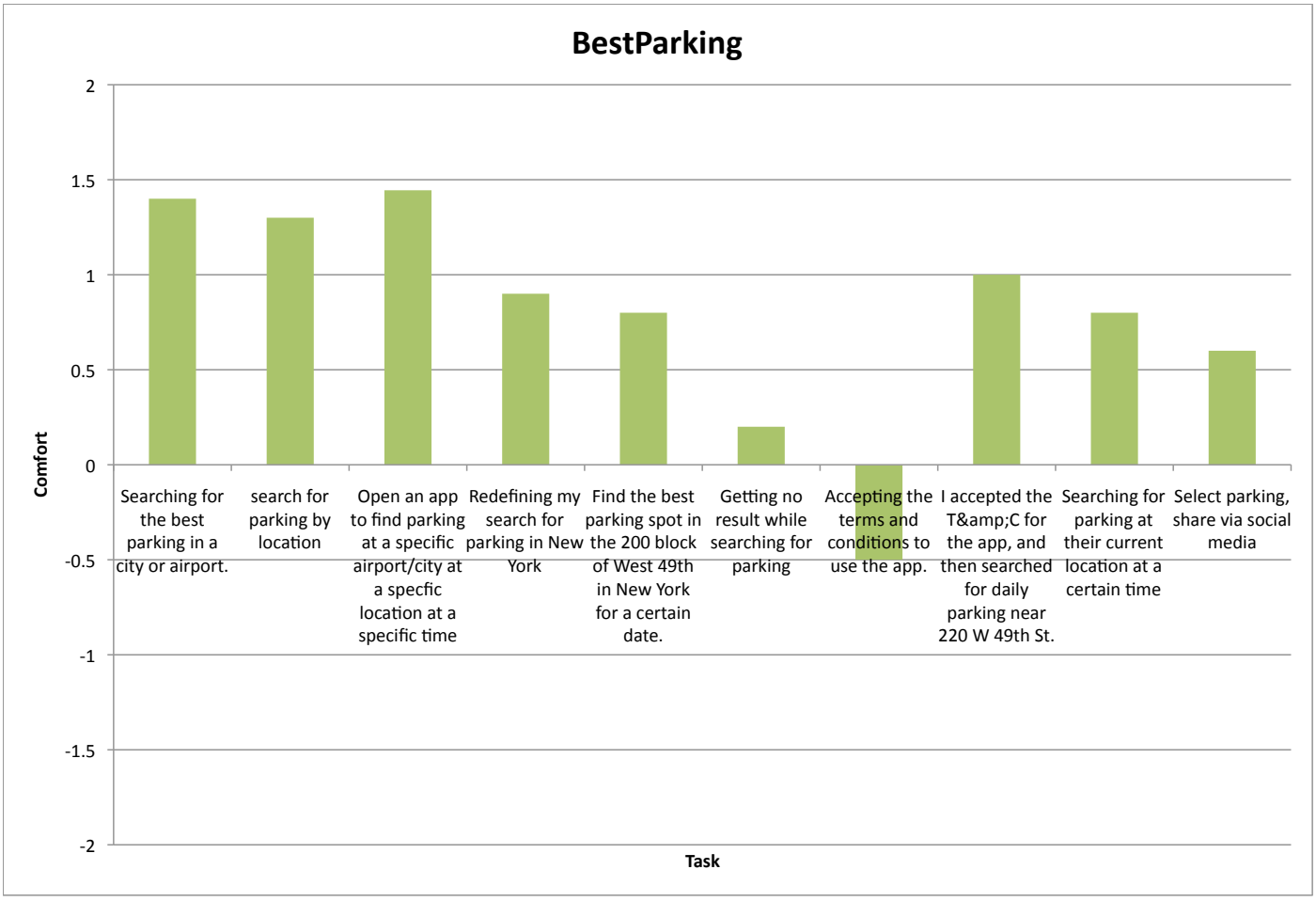
55) Any other feedback/comments?

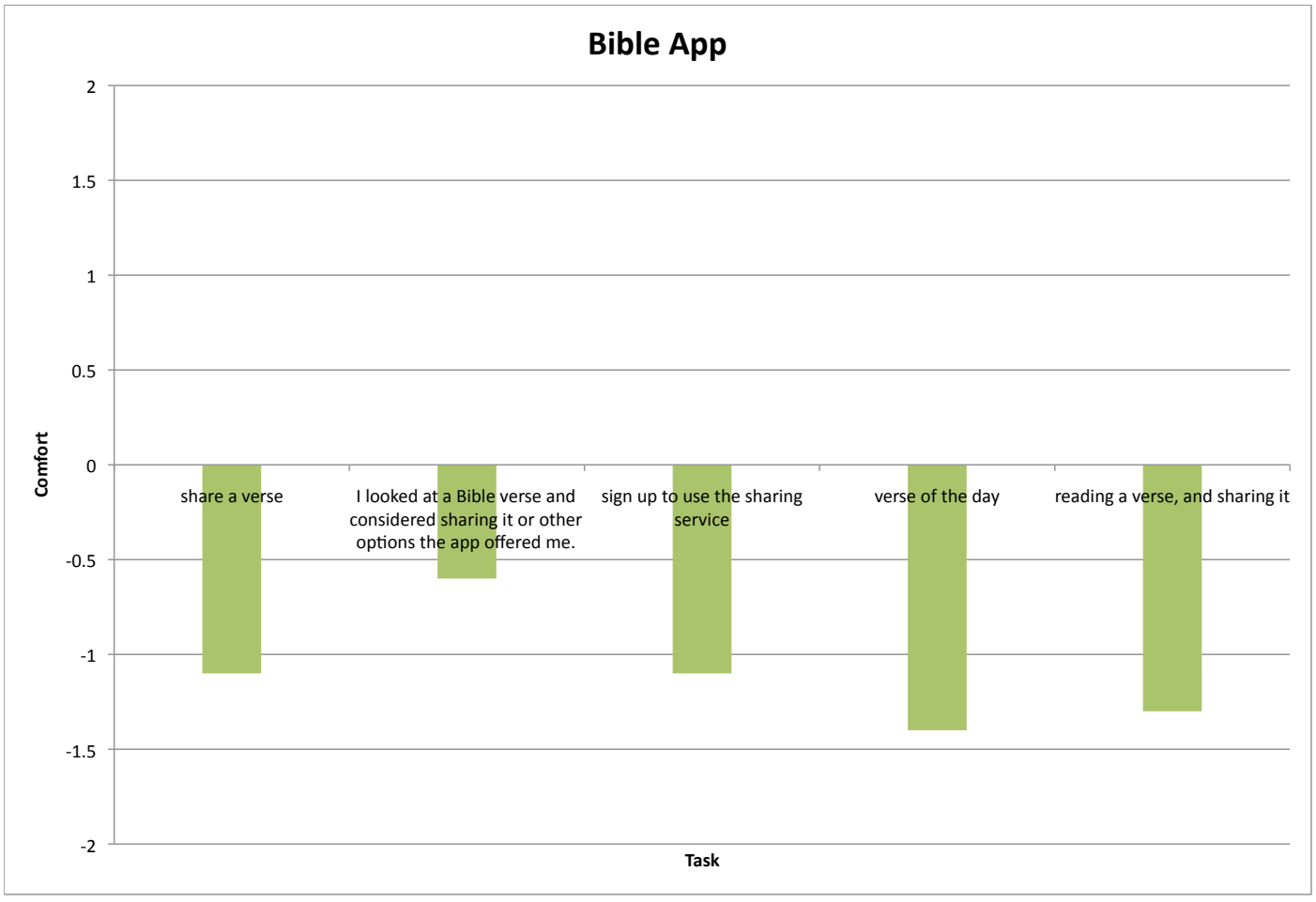
_____
_____
_____
_____

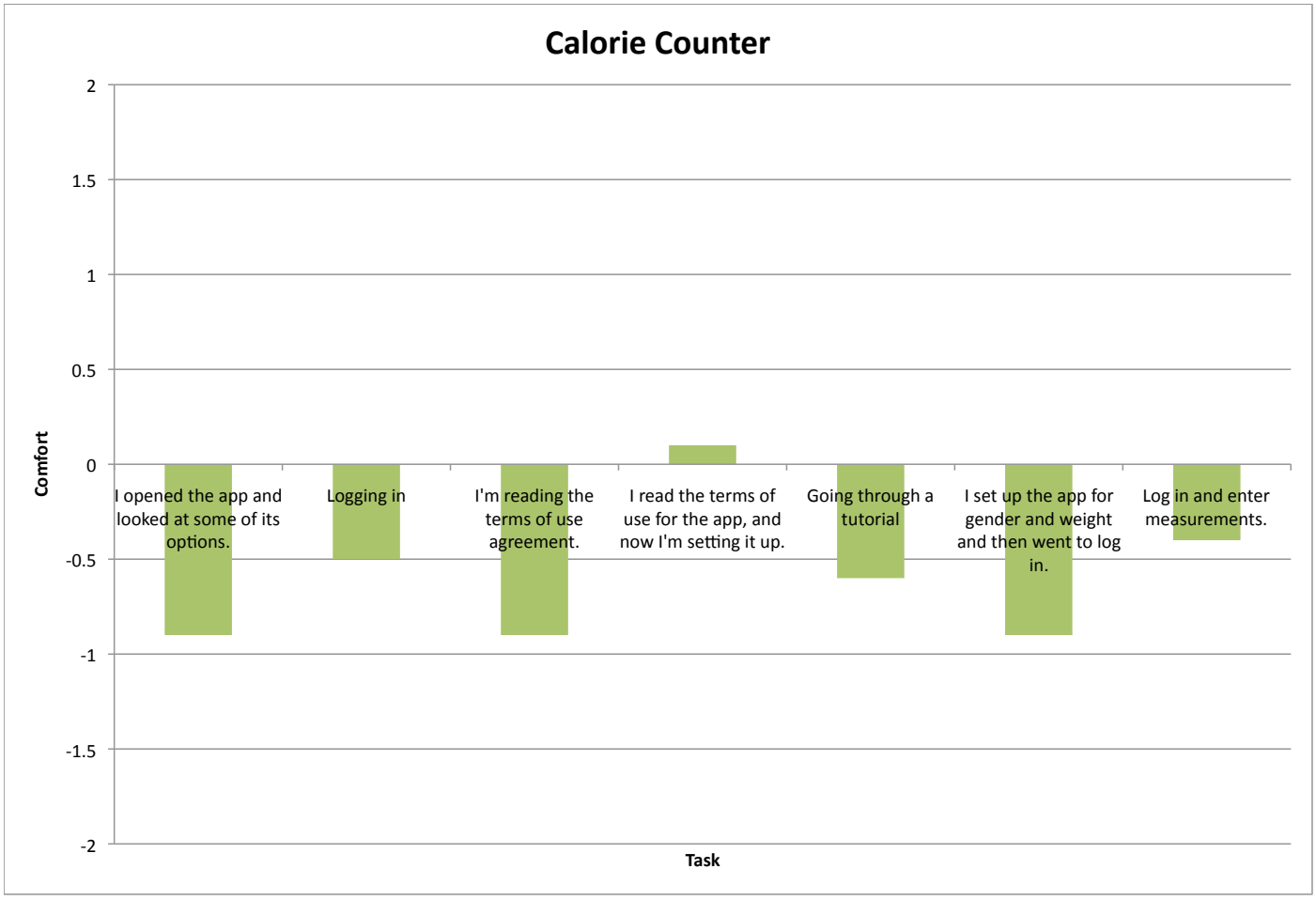
H Crowd App Analysis Charts

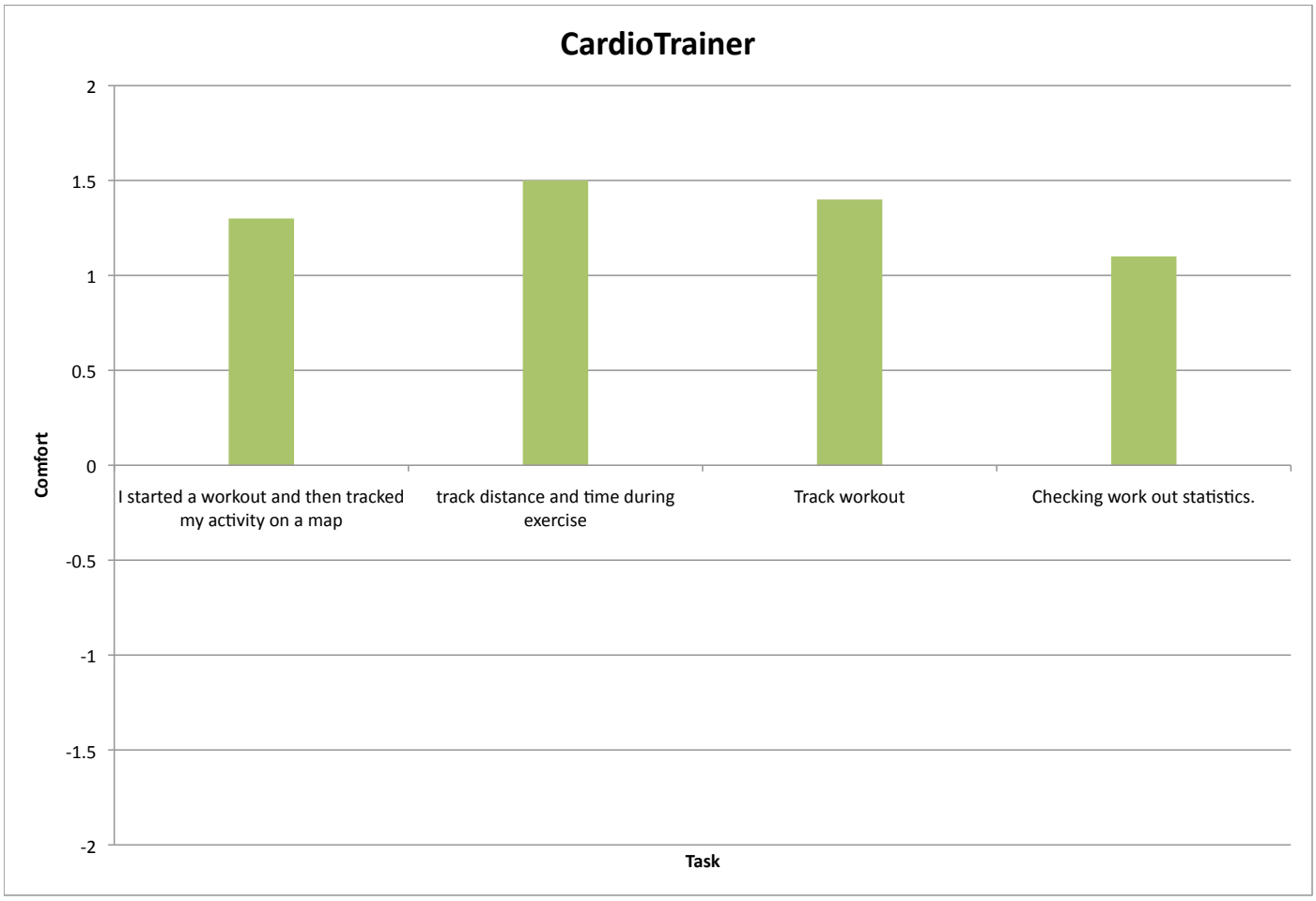
Appendix H presents the tasks extracted through Crowd Analysis and the associated crowd comfort levels for the 22 Android apps listed below. The apps were evaluated with Crowd Analysis after being automatically scanned by the traversal algorithm and processed by the Gort heuristics. For samples of the sensitive information transmitted by each app refer to Appendix I.

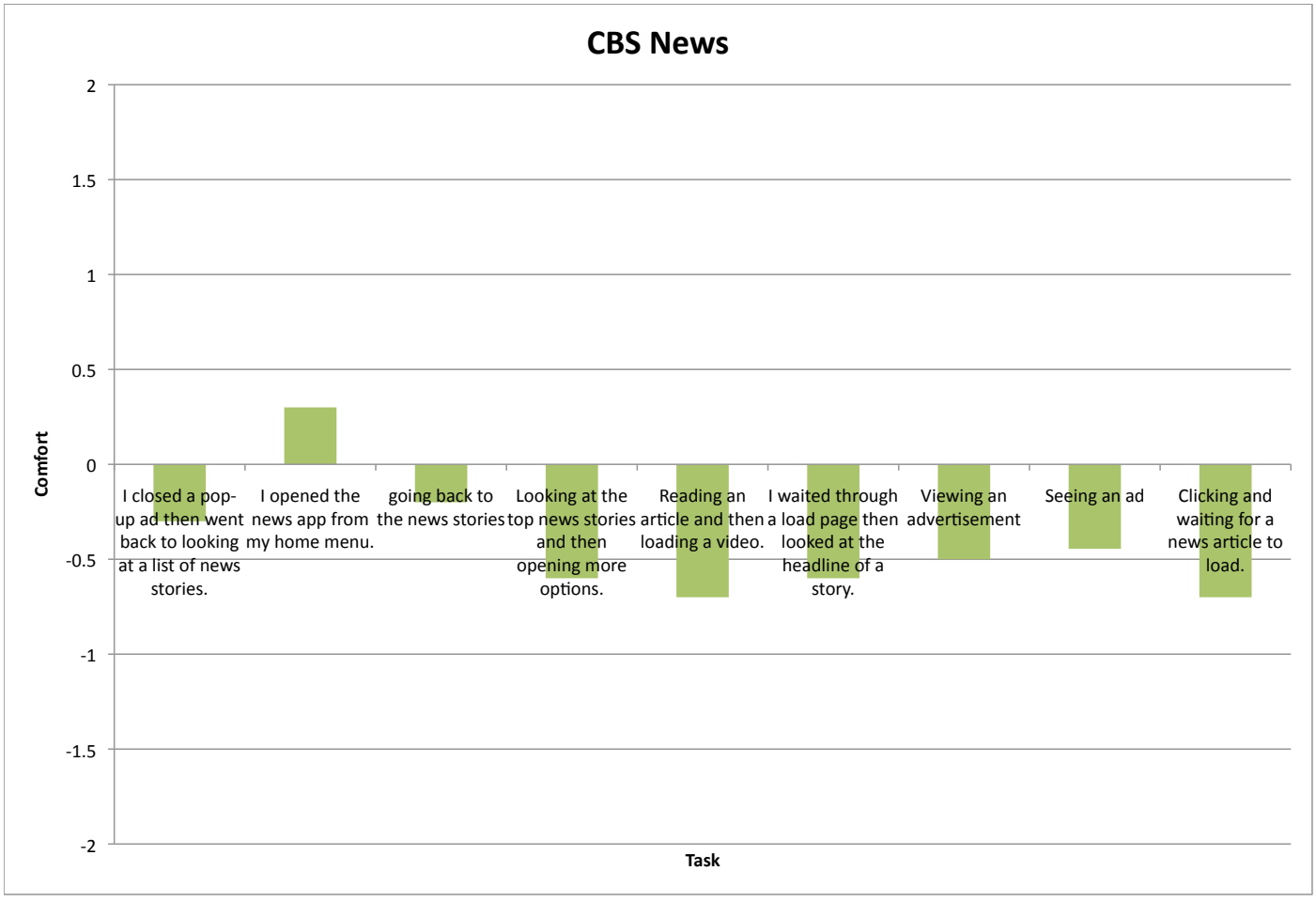
1. BestParking
2. Bible App
3. CalorieCounter
4. CardioTrainer
5. CBSNews
6. Fandango
7. Flashlight
8. GasBuddy
9. Groupon
10. Horoscope
11. iRadar
12. KBB.com
13. MapQuest
14. PhoneInfos
15. QRCodeReader
16. Shazam
17. Speedtest
18. TripAdvisor
19. WeatherBug
20. WhitePages
21. Yellow Pages
22. Yelp

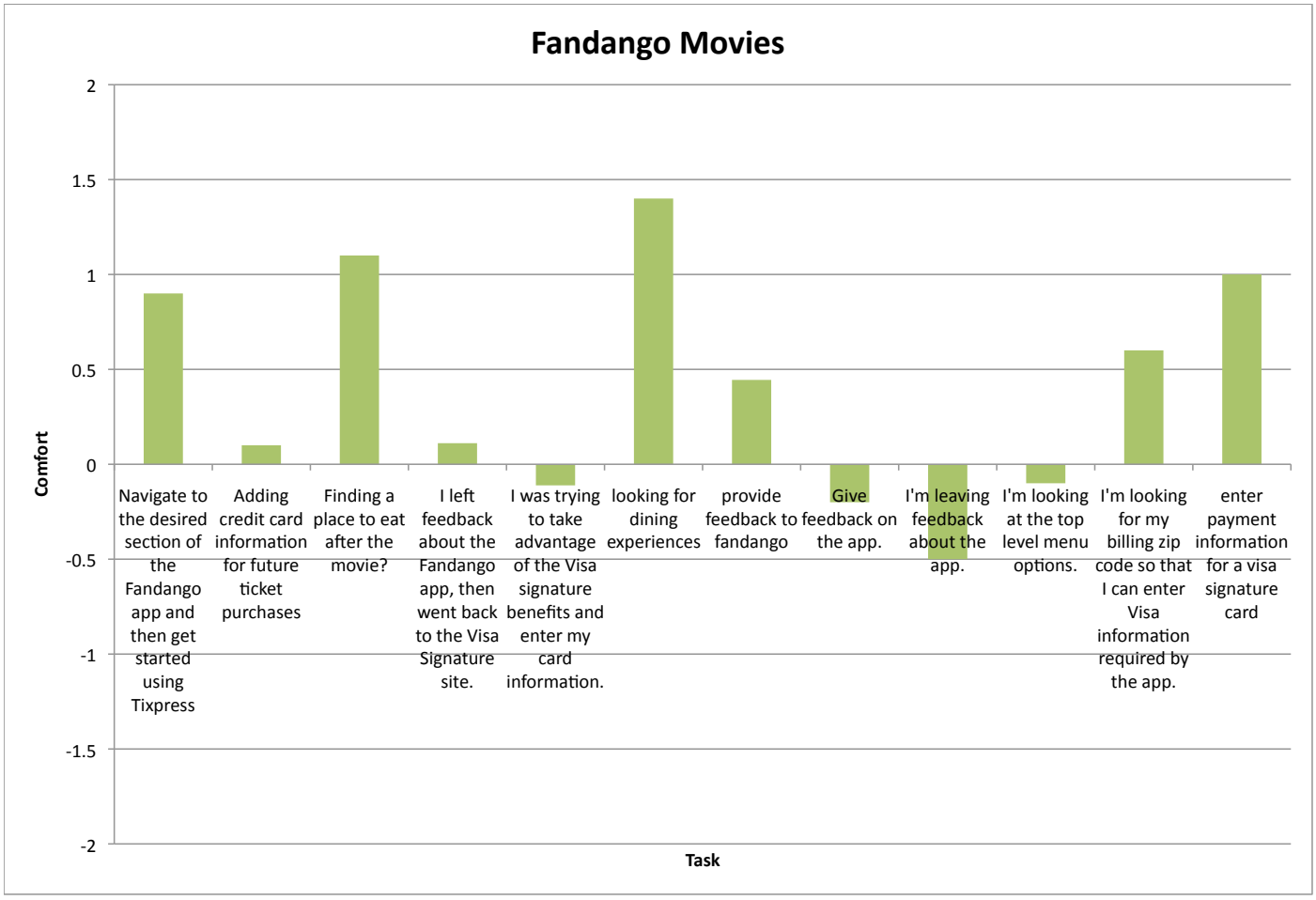

BestParking
Comfort
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
Searching for the best parking in a city or airport.
search for parking by location
Open an app to find parking at a specific airport/city at a specfic location at a specific time
Redefining my search for parking in New York
Find the best parking spot in the 200 block of West 49th in New York for a certain date.
Getting no result while searching for parking
Accepting the terms and conditions to use the app.
I accepted the T&C for the app, and then searched for daily parking near 220 W 49th St.
Searching for parking at their current location at a certain time
Select parking, share via social media
Task

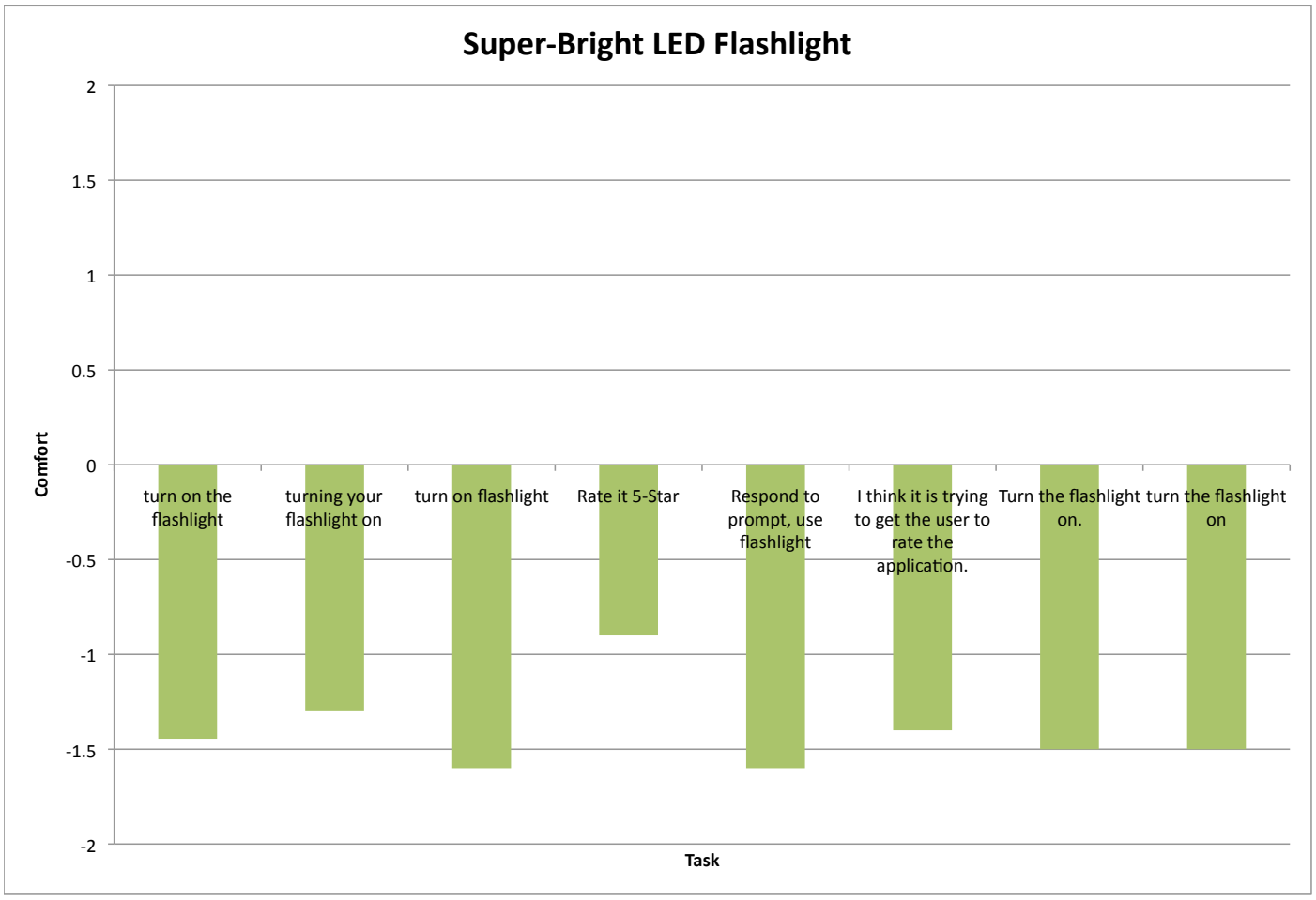

Bible App
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
Comfort
share a verse
I looked at a Bible verse and considered sharing it or other options the app offered me.
sign up to use the sharing service
verse of the day
reading a verse, and sharing it
Task

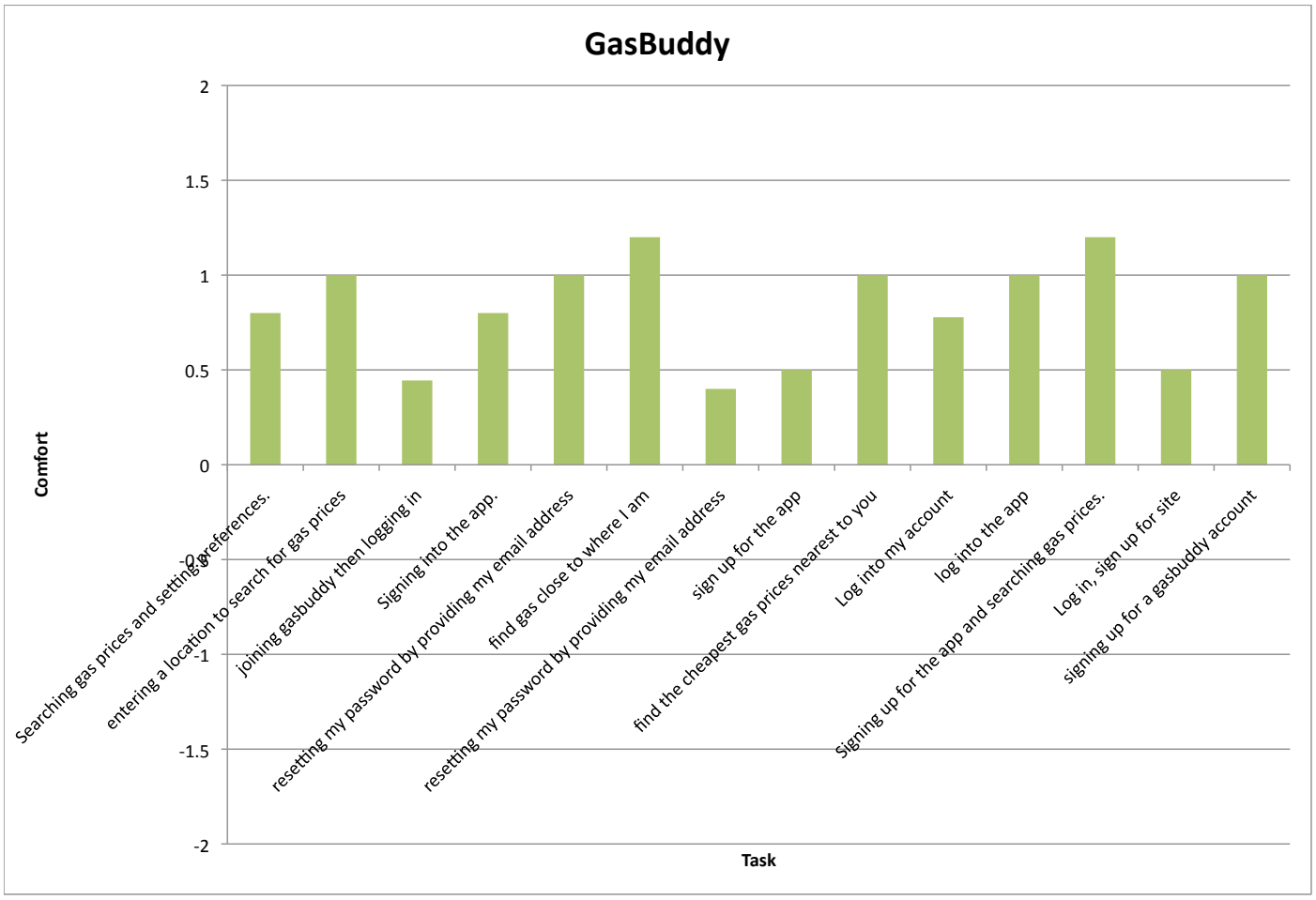

Calorie Counter
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
Comfort
I opened the app and looked at some of its options.
Logging in
I'm reading the terms of use agreement.
I read the terms of use for the app, and now I'm setting it up.
Going through a tutorial
I set up the app for gender and weight and then went to log in.
Log in and enter measurements.
Task

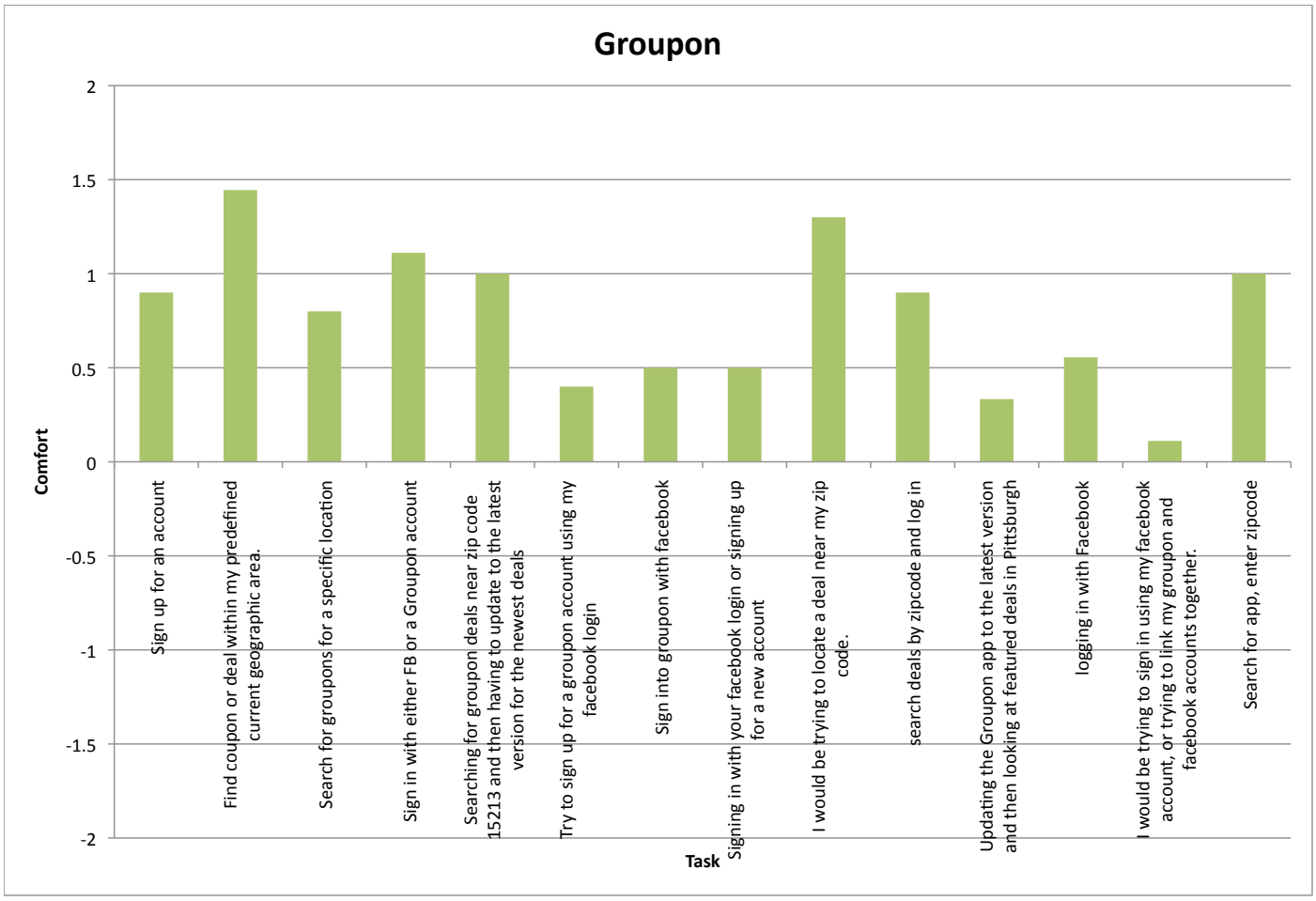

CardioTrainer
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
Comfort
I started a workout and then tracked my activity on a map
track distance and time during exercise
Track workout
Checking work out statistics.
Task

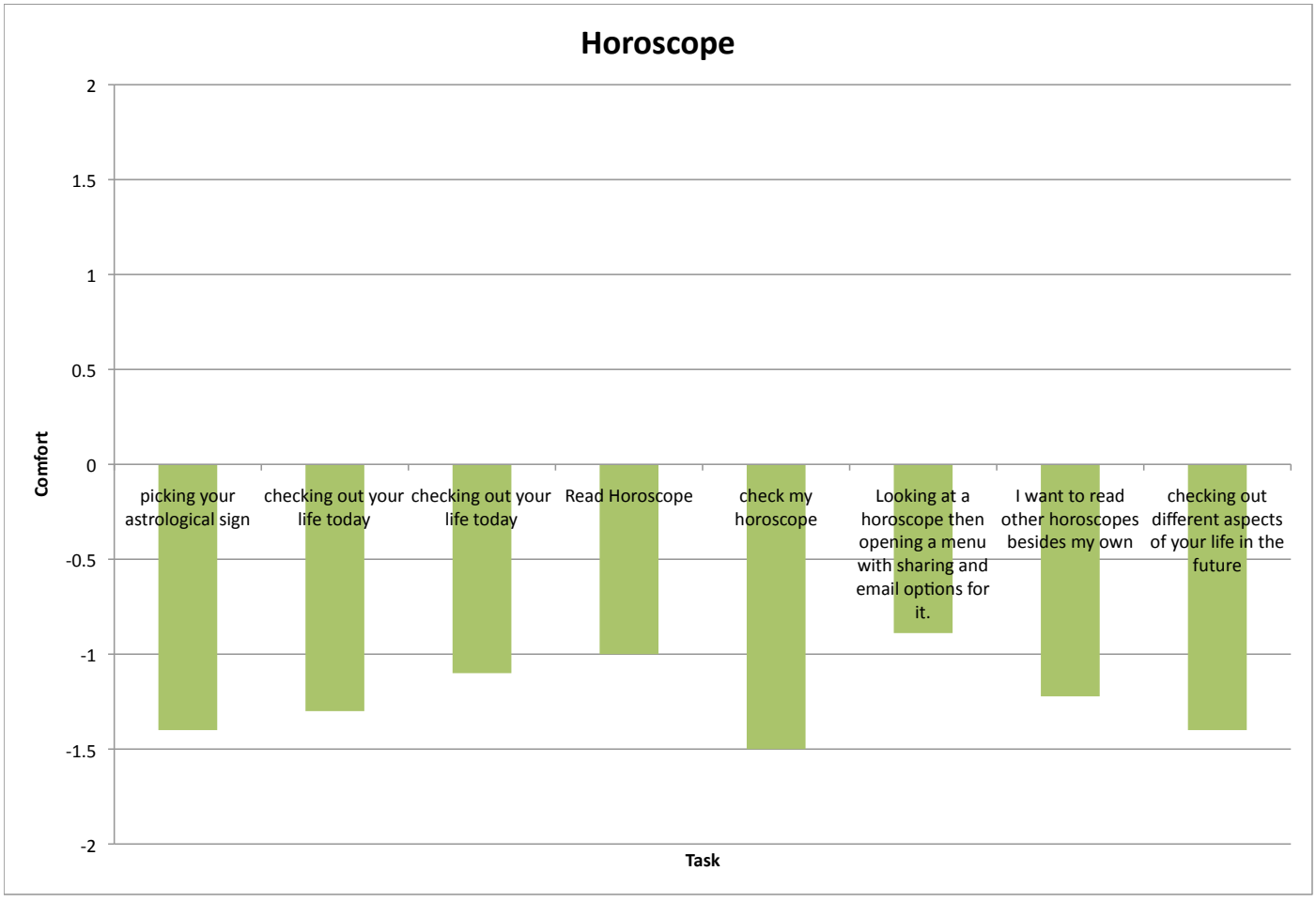

CBS News
Comfort
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
I closed a pop-up ad then went back to looking at a list of news stories.
I opened the news app from my home menu.
going back to the news stories
Looking at the top news stories and then opening more options.
Reading an article and then loading a video.
I waited through a load page then looked at the headline of a story.
Viewing an advertisement
Seeing an ad
Clicking and waiting for a news article to load.
Task

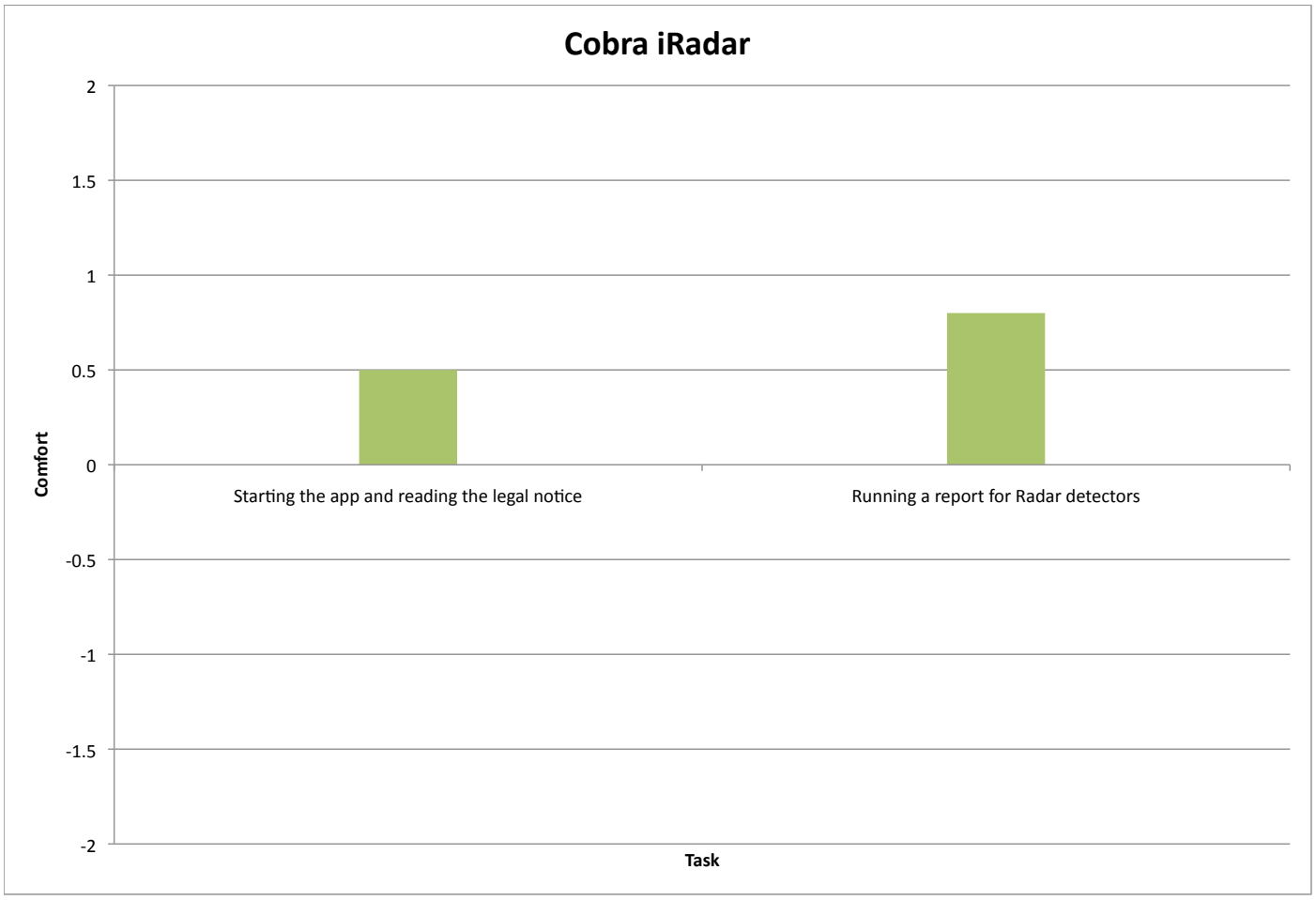

Fandango Movies
Comfort
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
Navigate to the desired section of the Fandango app and then get started using Tixpress
Adding credit card information for future ticket purchases
Finding a place to eat after the movie?
I left feedback about the Fandango app, then went back to the Visa Signature site.
I was trying to take advantage of the Visa signature benefits and enter my card information.
looking for dining experiences
provide feedback to fandango
Give feedback on the app.
I'm leaving feedback about the app.
I'm looking at the top level menu options.
I'm looking for my billing zip code so that I can enter Visa information required by the app.
enter payment information for a visa signature card
Task

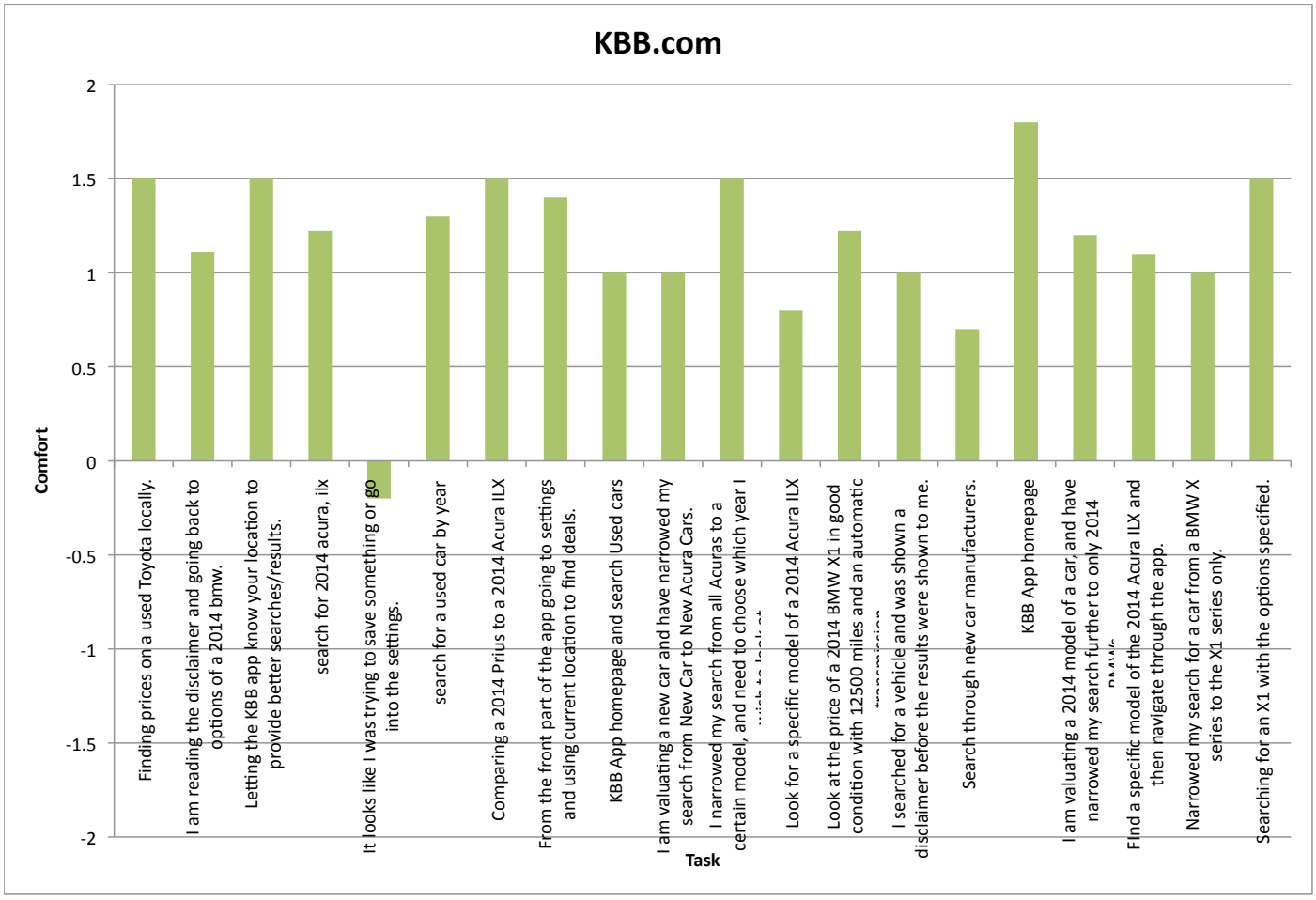

Super-Bright LED Flashlight
Comfort
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
turn on the flashlight
turning your flashlight on
turn on flashlight
Rate it 5-Star
Respond to prompt, use flashlight
I think it is trying to get the user to rate the application.
Turn the flashlight on.
turn the flashlight on
Task

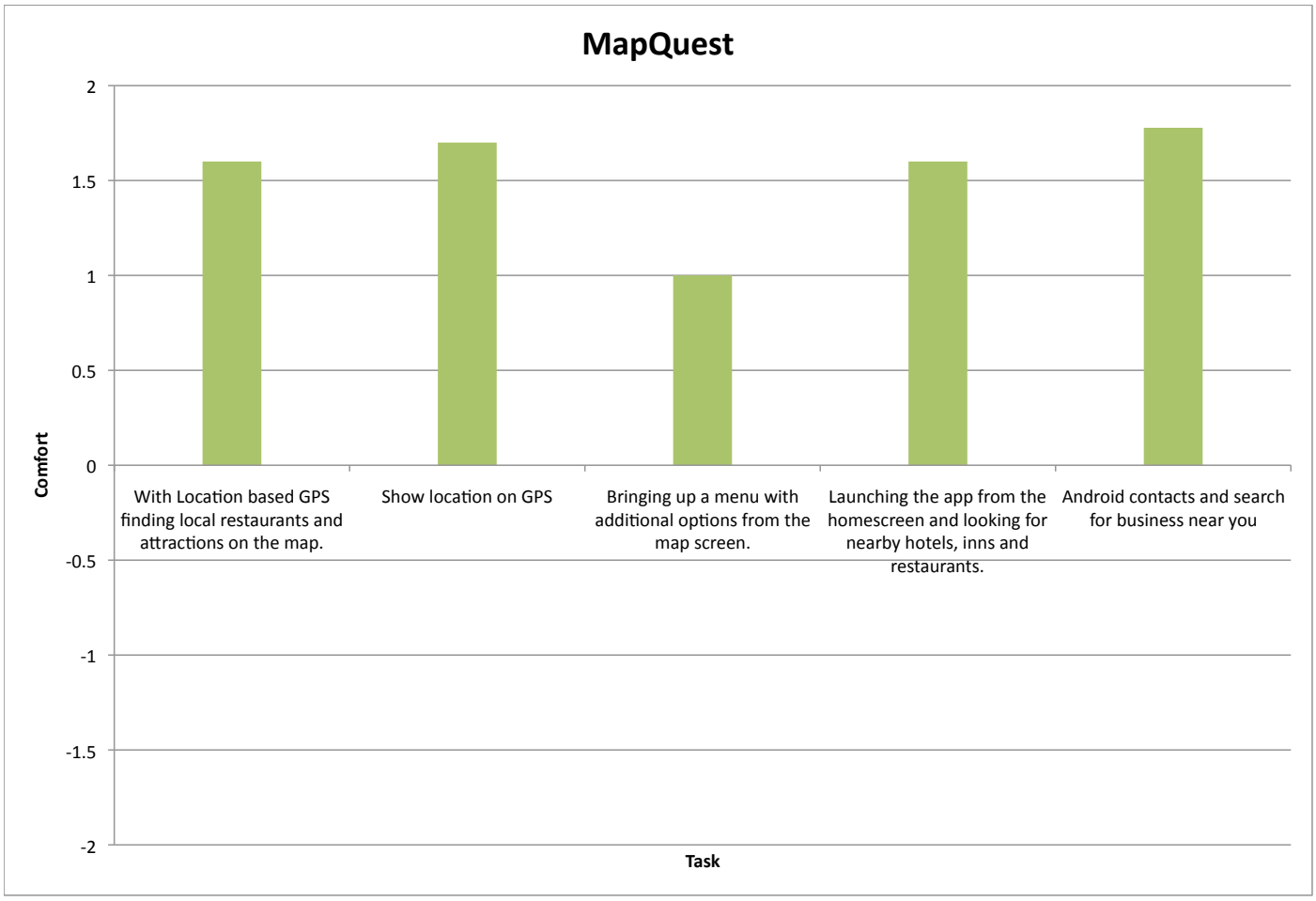

GasBuddy
Comfort
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
Searching gas prices and setting preferences.
entering a location to search for gas prices
joining gasbuddy then logging in
Signing into the app.
resetting my password by providing my email address
find gas close to where I am
resetting my password by providing my email address
sign up for the app
find the cheapest gas prices nearest to you
Log into my account
log into the app
Signing up for the app and searching gas prices.
Log in, sign up for site
signing up for a gasbuddy account
Task

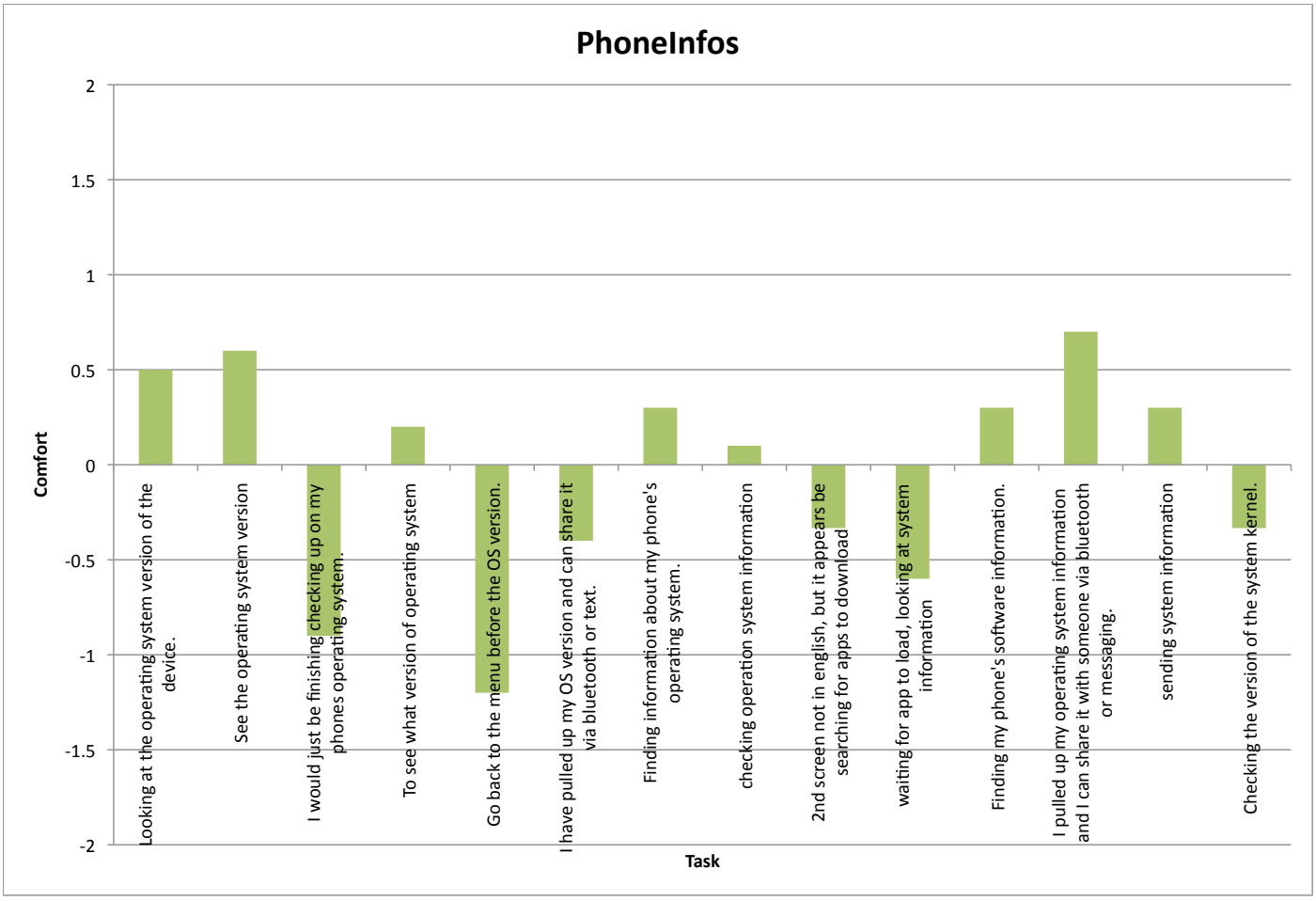

Groupon
Comfort
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
Sign up for an account
Find coupon or deal within my predefined current geographic area.
Search for groupons for a specific location
Sign in with either FB or a Groupon account
Searching for groupon deals near zip code 15213 and then having to update to the latest version for the newest deals
Try to sign up for a groupon account using my facebook login
Sign into groupon with facebook
Signing in with your facebook login or signing up for a new account
I would be trying to locate a deal near my zip code.
search deals by zipcode and log in
Updating the Groupon app to the latest version and then looking at featured deals in Pittsburgh
logging in with Facebook
I would be trying to sign in using my facebook account, or trying to link my groupon and facebook accounts together.
Search for app, enter zipcode
Task

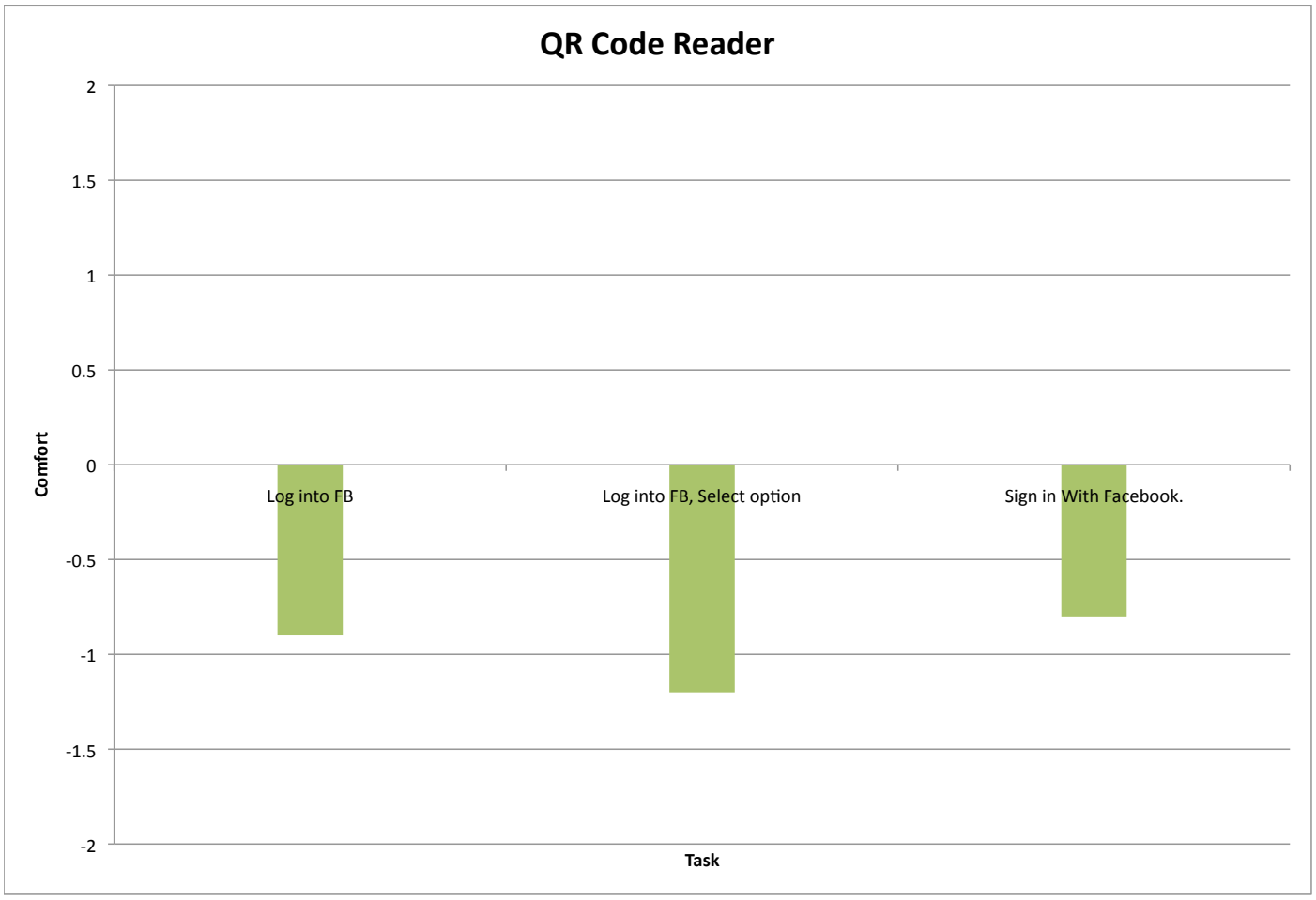

Horoscope
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
Comfort
picking your astrological sign
checking out your life today
checking out your life today
Read Horoscope
check my horoscope
Looking at a horoscope then opening a menu with sharing and email options for it.
I want to read other horoscopes besides my own
checking out different aspects of your life in the future
Task

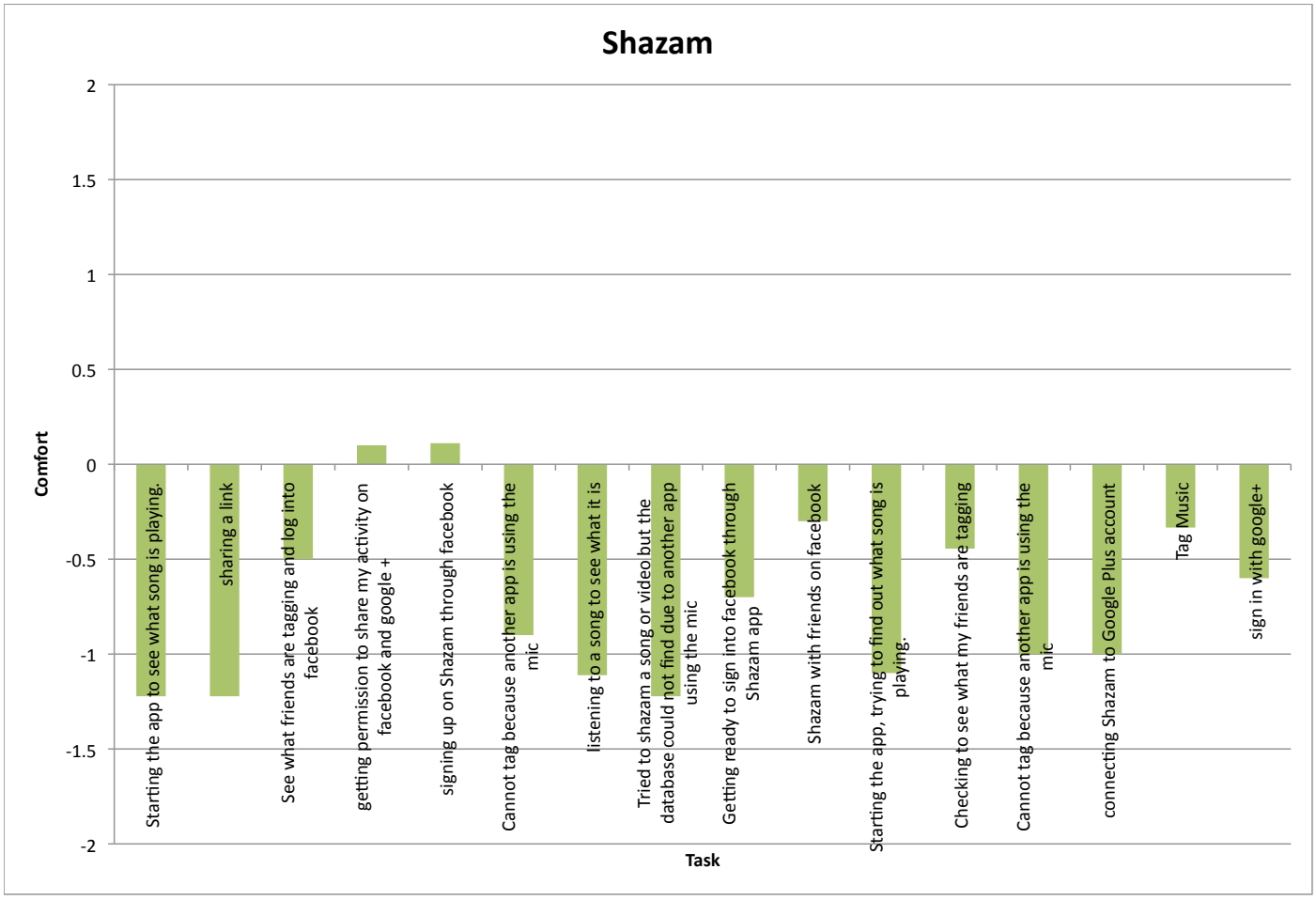

Cobra iRadar
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
Comfort
Starting the app and reading the legal notice
Running a report for Radar detectors
Task


KBB.com
Comfort
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
Finding prices on a used Toyota locally.
I am reading the disclaimer and going back to options of a 2014 bmw.
Letting the KBB app know your location to provide better searches/results.
search for 2014 acura, ilx
It looks like I was trying to save something or go into the settings.
search for a used car by year
Comparing a 2014 Prius to a 2014 Acura ILX
From the front part of the app going to settings and using current location to find deals.
KBB App homepage and search Used cars
I am valuating a new car and have narrowed my search from New Car to New Acura Cars.
I narrowed my search from all Acuras to a certain model, and need to choose which year I
Look for a specific model of a 2014 Acura ILX
Look at the price of a 2014 BMW X1 in good condition with 12500 miles and an automatic
I searched for a vehicle and was shown a disclaimer before the results were shown to me.
Search through new car manufacturers.
KBB App homepage
I am valuating a 2014 model of a car, and have narrowed my search further to only 2014
Find a specific model of the 2014 Acura ILX and then navigate through the app.
Narrowed my search for a car from a BMW X series to the X1 series only.
Searching for an X1 with the options specified.
Task

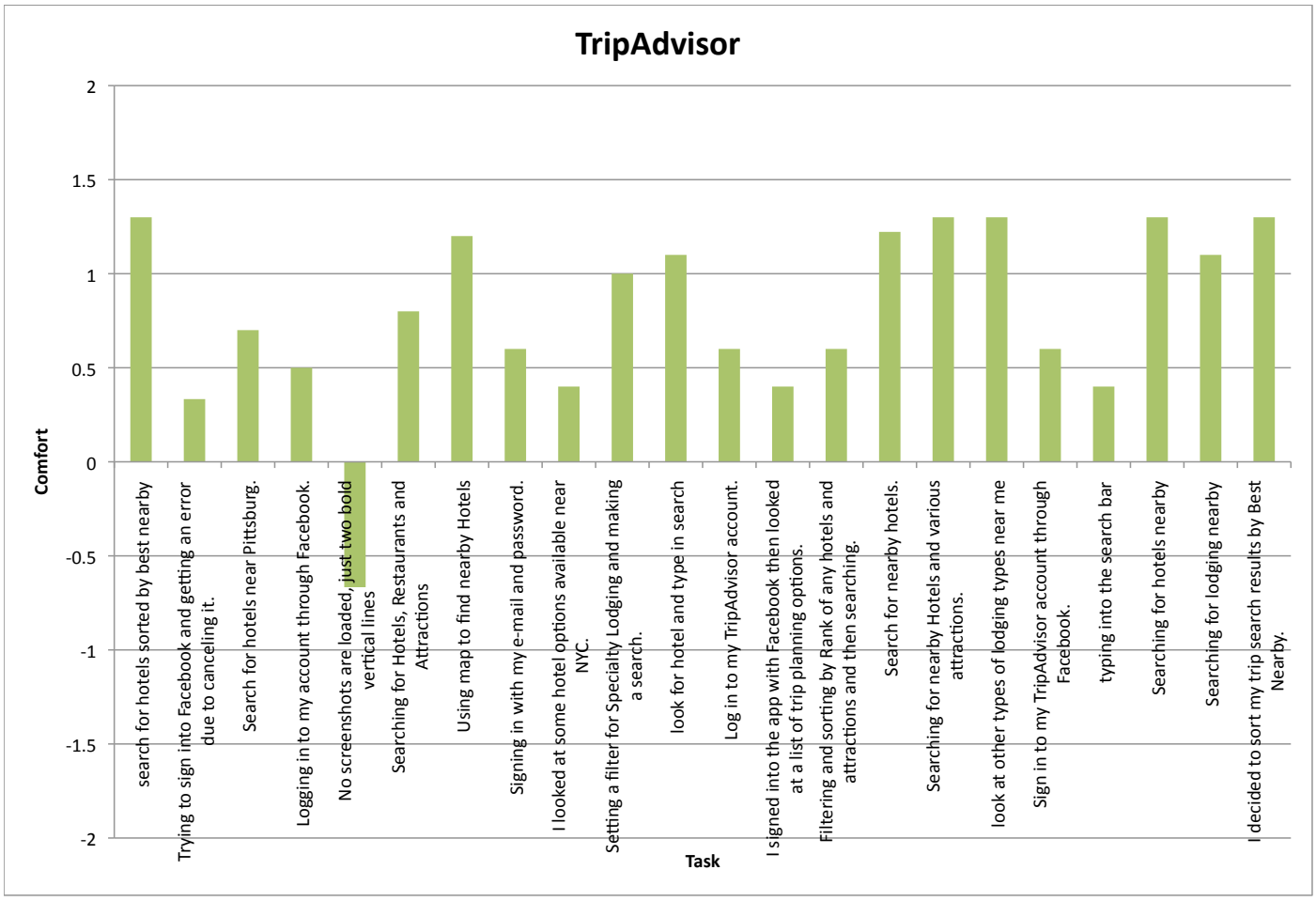

MapQuest
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
Comfort
With Location based GPS finding local restaurants and attractions on the map.
Show location on GPS
Bringing up a menu with additional options from the map screen.
Launching the app from the homescreen and looking for nearby hotels, inns and restaurants.
Android contacts and search for business near you
Task

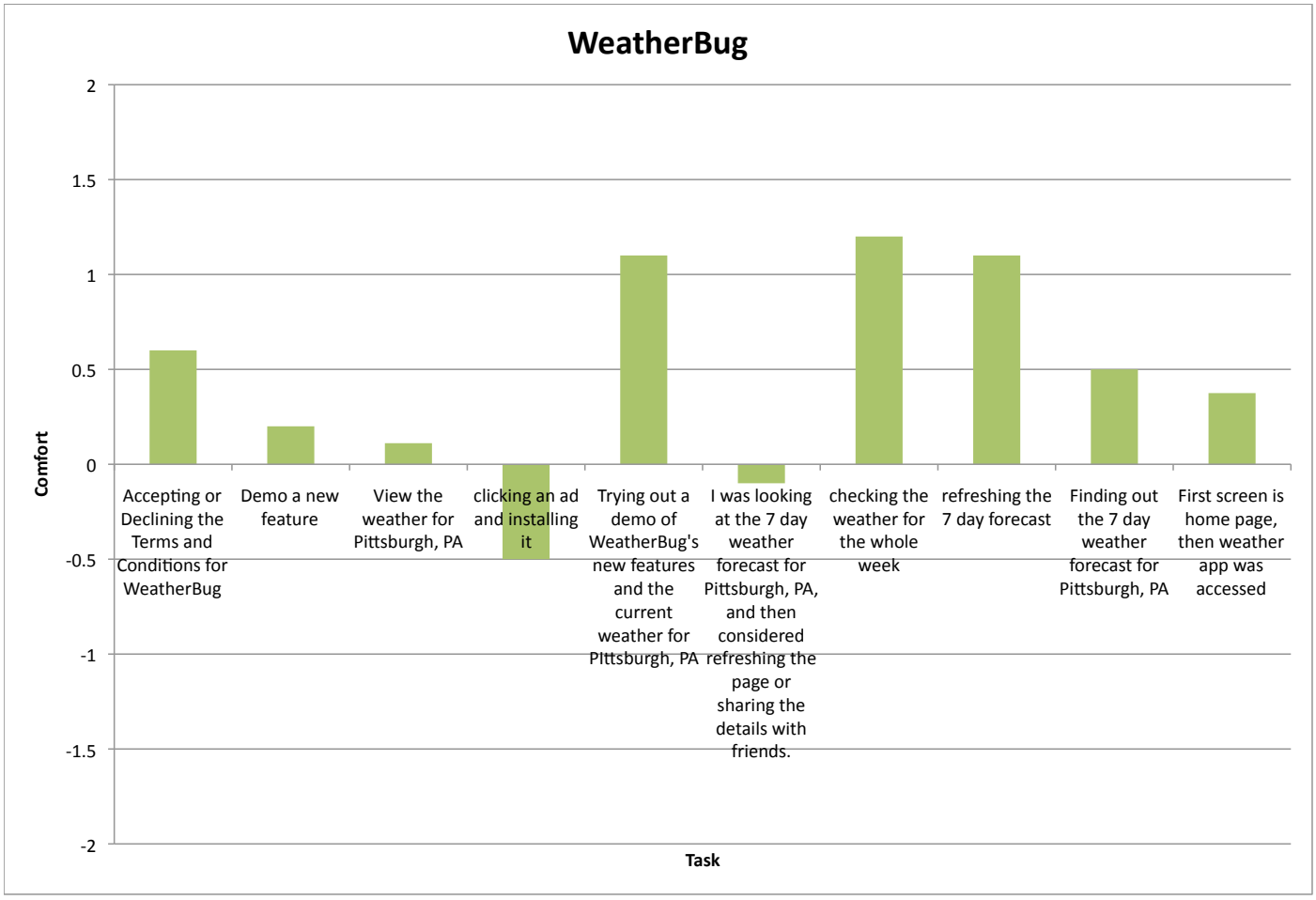

PhoneInfos
Comfort
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
Looking at the operating system version of the device.
See the operating system version
I would just be finishing checking up on my phones operating system.
To see what version of operating system
Go back to the menu before the OS version.
I have pulled up my OS version and can share it via bluetooth or text.
Finding information about my phone's operating system.
checking operation system information
2nd screen not in english, but it appears be searching for apps to download
waiting for app to load, looking at system information
Finding my phone's software information.
I pulled up my operating system information and I can share it with someone via bluetooth or messaging.
sending system information
Checking the version of the system kernel.
Task

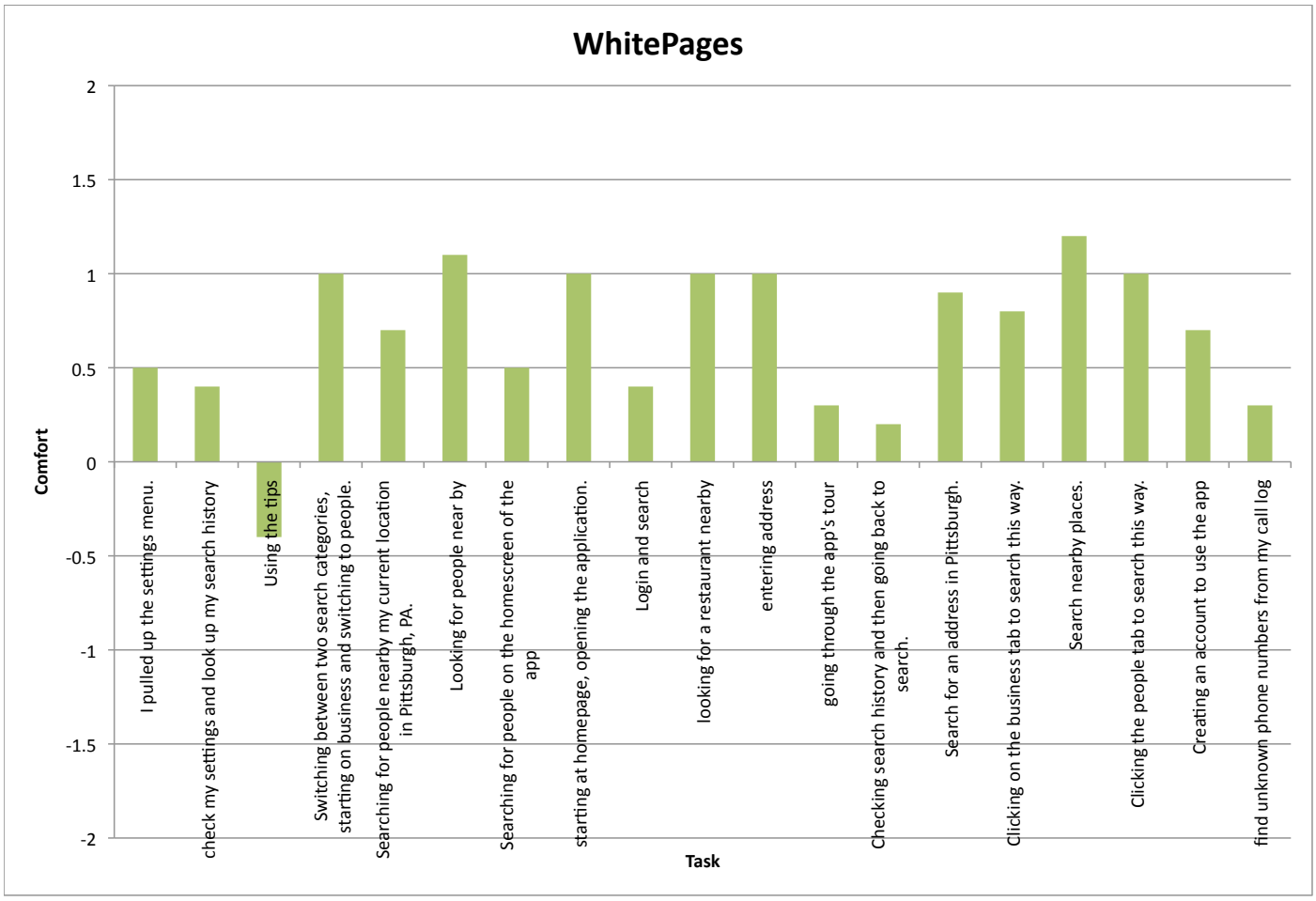

QR Code Reader
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
Comfort
Log into FB
Log into FB, Select option
Sign in With Facebook.
Task

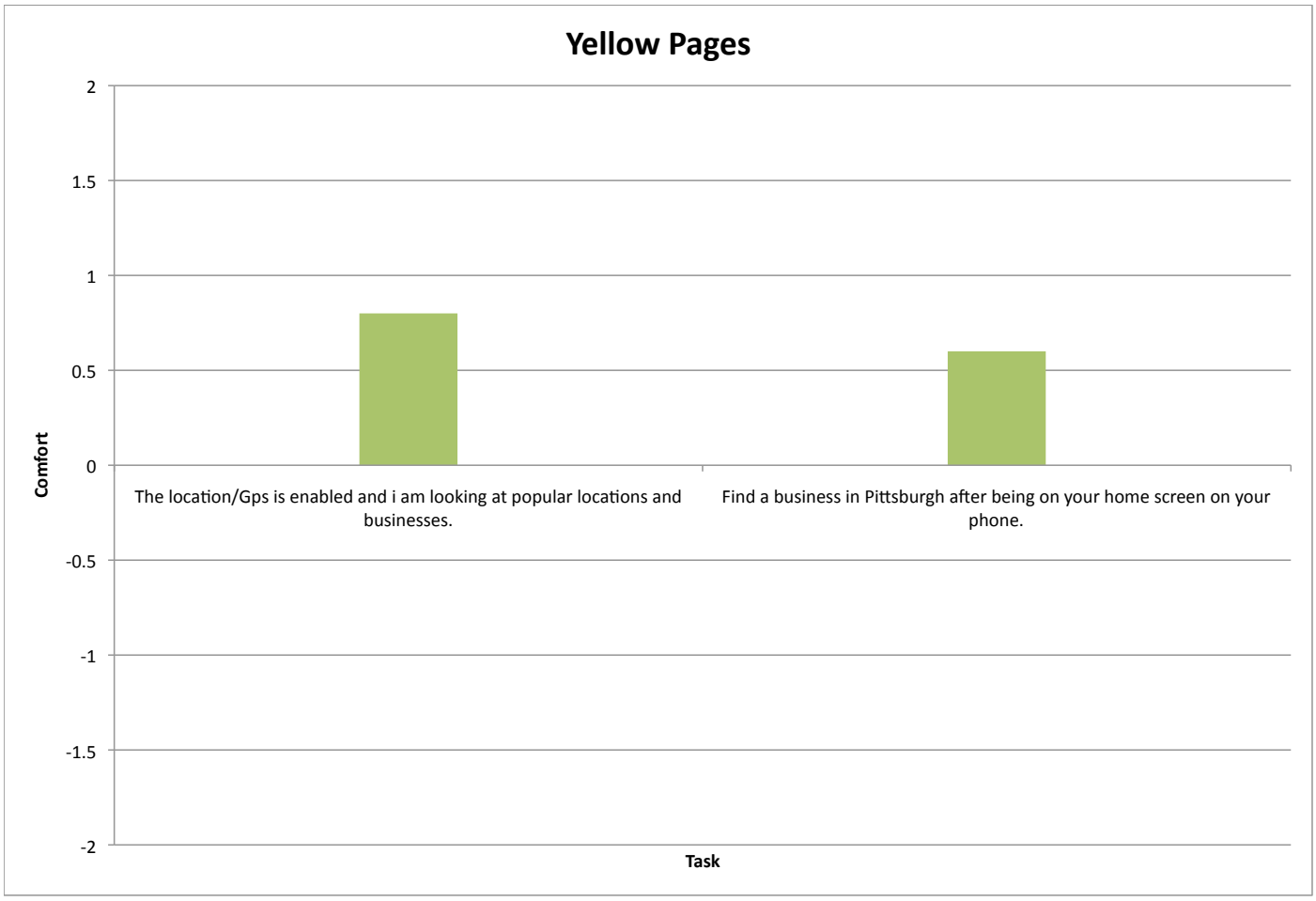

Shazam
Comfort
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
Starting the app to see what song is playing.
sharing a link
See what friends are tagging and log into facebook
getting permission to share my activity on facebook and google +
signing up on Shazam through facebook
Cannot tag because another app is using the mic
listening to a song to see what it is
Tried to shazam a song or video but the database could not find due to another app using the mic
Getting ready to sign into facebook through Shazam app
Shazam with friends on facebook
Starting the app, trying to find out what song is playing.
Checking to see what my friends are tagging
Cannot tag because another app is using the mic
connecting Shazam to Google Plus account
Tag Music
sign in with google+
Task

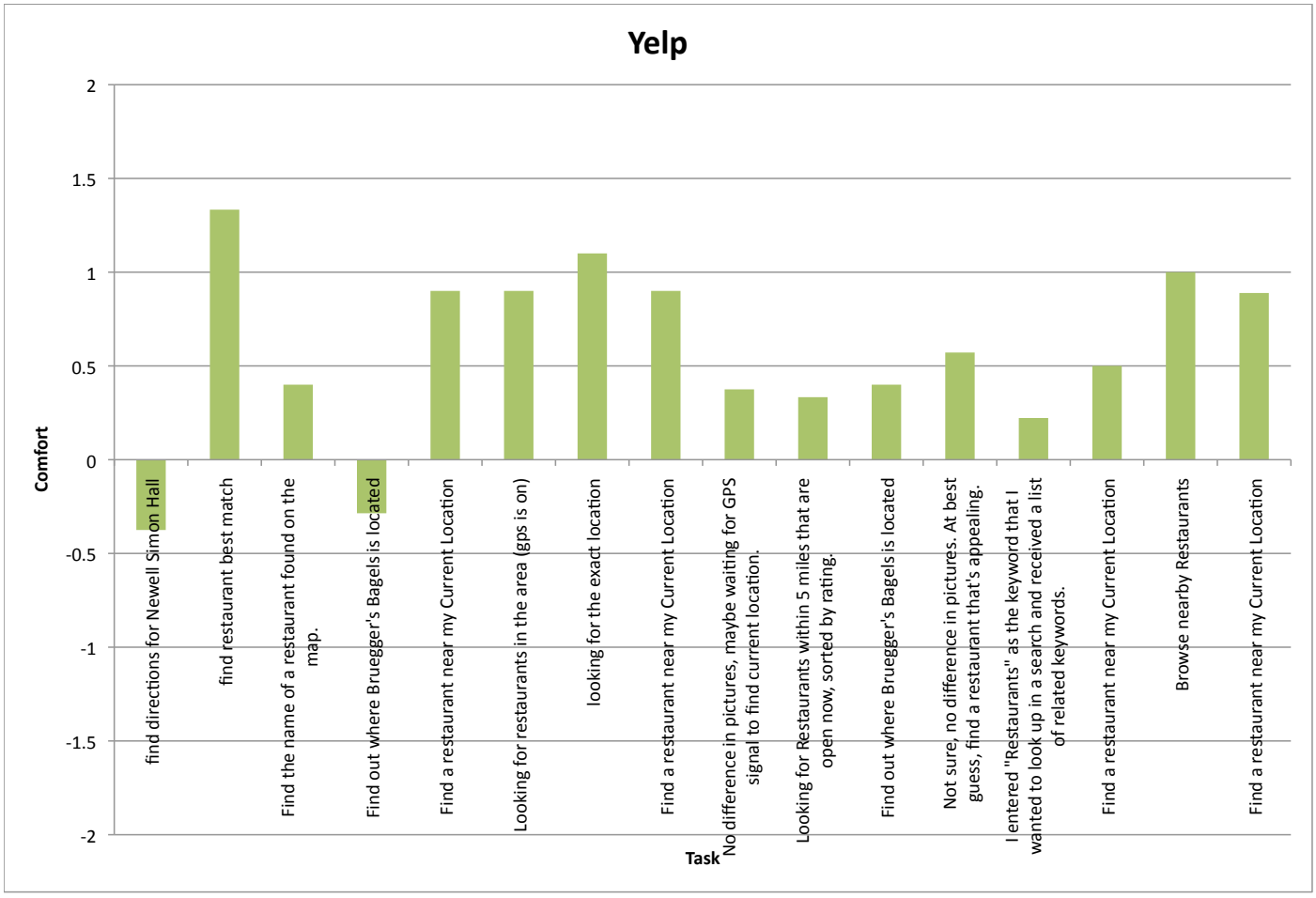

Speedtest.net
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
Comfort
I am trying to leave feedback
Respond to prompt, begin test
Task


TripAdvisor
Comfort
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
search for hotels sorted by best nearby
Trying to sign into Facebook and getting an error due to canceling it.
Search for hotels near Pittsburg.
Logging in to my account through Facebook.
No screenshots are loaded, just two bold vertical lines
Searching for Hotels, Restaurants and Attractions
Using map to find nearby Hotels
Signing in with my e-mail and password.
I looked at some hotel options available near NYC.
Setting a filter for Specialty Lodging and making a search.
look for hotel and type in search
Log in to my TripAdvisor account.
I signed into the app with Facebook then looked at a list of trip planning options.
Filtering and sorting by Rank of any hotels and attractions and then searching.
Search for nearby hotels.
Searching for nearby Hotels and various attractions.
look at other types of lodging types near me
Sign in to my TripAdvisor account through Facebook.
typing into the search bar
Searching for hotels nearby
Searching for lodging nearby
I decided to sort my trip search results by Best Nearby.
Task


WeatherBug
Comfort
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
Accepting or Declining the Terms and Conditions for WeatherBug
Demo a new feature
View the weather for Pittsburgh, PA
clicking an ad and installing it
Trying out a demo of WeatherBug's new features and the current weather for Pittsburgh, PA
I was looking at the 7 day weather forecast for Pittsburgh, PA, and then considered refreshing the page or sharing the details with friends.
checking the weather for the whole week
refreshing the 7 day forecast
Finding out the 7 day weather forecast for Pittsburgh, PA
First screen is home page, then weather app was accessed
Task


WhitePages
Comfort
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
I pulled up the settings menu.
check my settings and look up my search history
Using the tips
Switching between two search categories, starting on business and switching to people.
Searching for people nearby my current location in Pittsburgh, PA.
Looking for people near by
Searching for people on the homescreen of the app
starting at homepage, opening the application.
Login and search
looking for a restaurant nearby
entering address
going through the app's tour
Checking search history and then going back to search.
Search for an address in Pittsburgh.
Clicking on the business tab to search this way.
Search nearby places.
Clicking the people tab to search this way.
Creating an account to use the app
find unknown phone numbers from my call log
Task


Yellow Pages
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
Comfort
The location/Gps is enabled and i am looking at popular locations and businesses.
Find a business in Pittsburgh after being on your home screen on your phone.
Task


Yelp
Comfort
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
find directions for Newell Simon Hall
find restaurant best match
Find the name of a restaurant found on the map.
Find out where Bruegger's Bagels is located
Find a restaurant near my Current Location
Looking for restaurants in the area (gps is on)
looking for the exact location
Find a restaurant near my Current Location
No difference in pictures, maybe waiting for GPS signal to find current location.
Looking for Restaurants within 5 miles that are open now, sorted by rating.
Find out where Bruegger's Bagels is located
Not sure, no difference in pictures. At best guess, find a restaurant that's appealing.
I entered "Restaurants" as the keyword that I wanted to look up in a search and received a list of related keywords.
Find a restaurant near my Current Location
Browse nearby Restaurants
Find a restaurant near my Current Location
Task

I Crowd App Analysis Results

This appendix presents the Crowd Analysis result samples for 22 Android apps. Crowd workers were shown a single screenshot in cases where the two screenshots are the same or very similar. The following apps were evaluated with Crowd Analysis after being automatically scanned by the traversal algorithm and processed by the Gort heuristics:

1. BestParking
2. Bible App
3. CalorieCounter
4. CardioTrainer
5. CBSNews
6. Fandango
7. Flashlight
8. GasBuddy
9. Groupon
10. Horoscope
11. iRadar
12. KBB.com
13. MapQuest
14. PhoneInfos
15. QRCodeReader
16. Shazam
17. Speedtest
18. TripAdvisor
19. WeatherBug
20. WhitePages
21. Yellow Pages
22. Yelp

Selected Task Label:

Accepting the terms and conditions to use the app.

Resources Used:

phone number, exact location, approximate location, and unique device identifier

Crowd expected resource use:

No

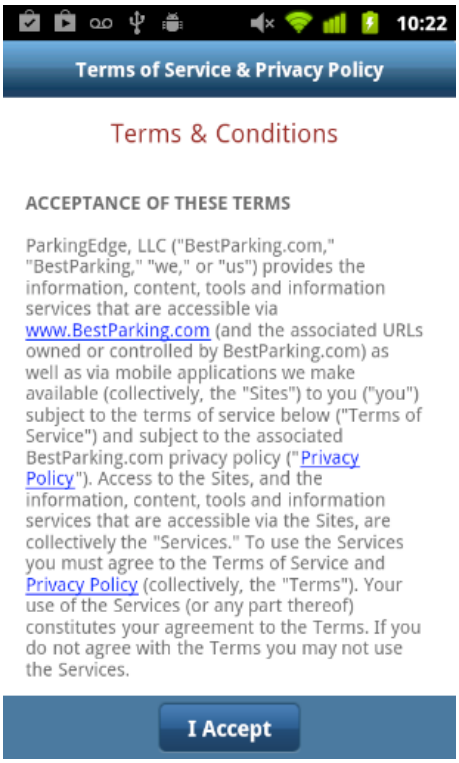
Crowd comfort [-2,2] (N, (μ,σ)):

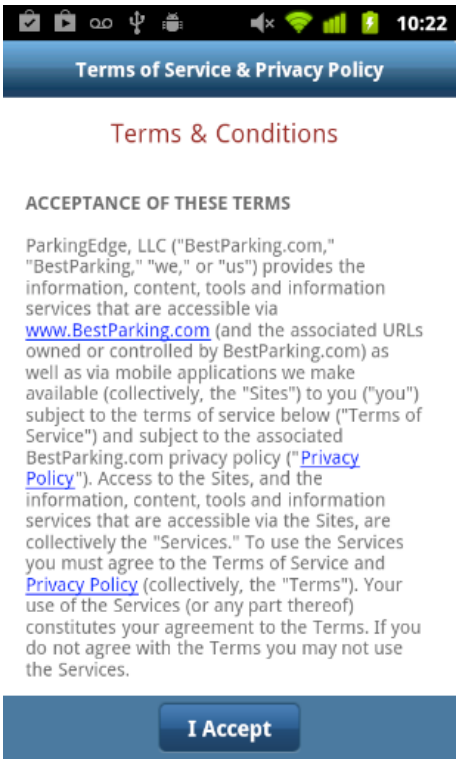
N=10, (μ=-0.5, σ=1.3)

Crowd-sourced Task Labels:

* Agree to the terms of service
* Agreeing to the Terms of Service.
* Find the cheapest parking
* accept the terms to use the app
* Accepting the Terms & Conditions
* accepting the terms and condtions
* Accept ToS
* agreeing to accept ToS and PP
* Accepting the terms and conditions to use the app.
* Accept ToS

Application Transition:





Selected Task Label:

Open an app to find parking at a specific airport/city at a specfic location at a specific time

Resources Used:

phone number, exact location, approximate location, and unique device identifier

Crowd expected resource use:

Yes

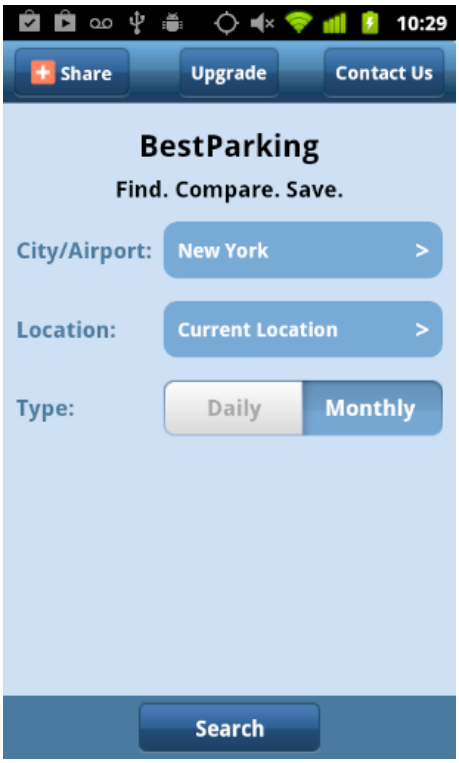
Crowd comfort [-2,2] (N, (μ,σ)):

N=9, (μ=1.4, σ=0.5)

Crowd-sourced Task Labels:

* Searching for best parking.
* looking for the best parking in the area
* Open an app to find parking at a specific airport/city at a specfic location at a specific time
* find best parking in ny monthly rates
* Open app from HS, select parking
* Searching the current location for a monthly parking space.
* The first screen looks to be a navigation screen, and the second is the app looking for the best parking
* finding a parking spot or garage, answering a phone call
* Launch app through homescreen, find parking
* launch app

Application Transition:



Selected Task Label:

verse of the day

Resources Used:

phone number, exact location, approximate location, and unique device identifier

Crowd expected resource use:

No

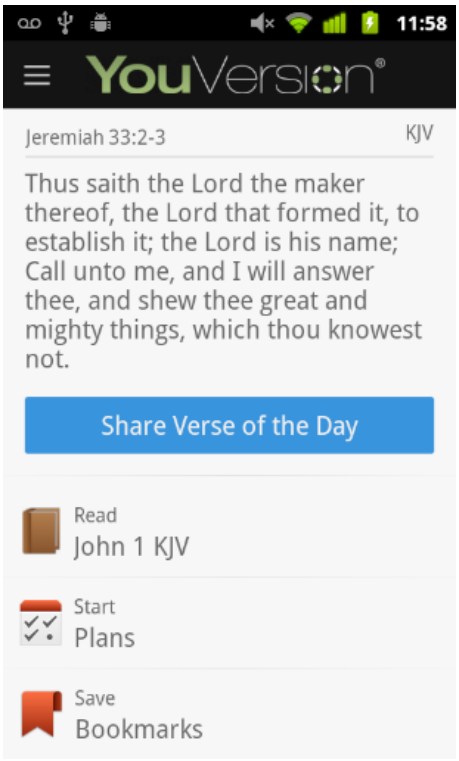
Crowd comfort [-2,2] (N, (μ,σ)):

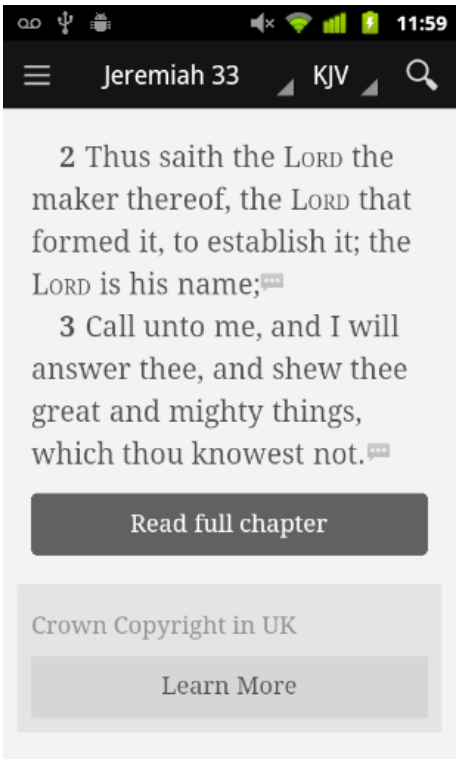
N=10, (μ=-1.4, σ=0.9)

Crowd-sourced Task Labels:

* verse of the day
* sharing the verse of the day with friends and reading passages
* reading the bible
* Share verse and read chapter
* Read part of the bible
* Find the specific verses.
* Find more information on a particular verse
* read a verse
* Read verse, continue reading bible
* I looked at a Biblical verse, and then explored the Bible chapter it was selected from.

Application Transition:





Selected Task Label:

I looked at a Bible verse and considered sharing it or other options the app offered me.

Resources Used:

phone number, exact location, approximate location, and unique device identifier

Crowd expected resource use:

No

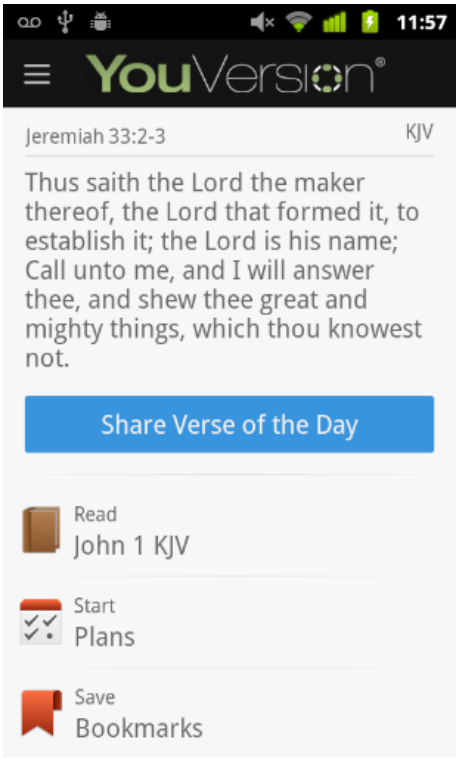
Crowd comfort [-2,2] (N, (μ,σ)):

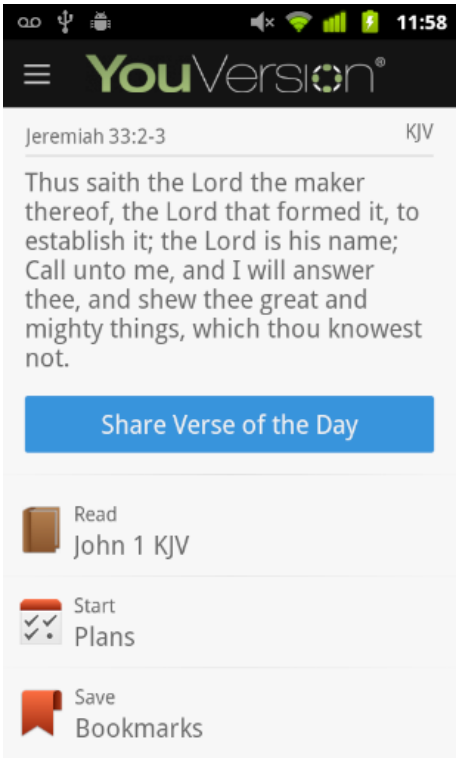
N=10, (μ=-0.6, σ=1.1)

Crowd-sourced Task Labels:

* Sharing verse of the day or book marking
* reading the bible
* reading a verse with an option of sharing it with friends
* share verse of the day
* Read daily verse for inspiration
* share a verse
* Read verse
* Read verse
* reading a verse and possibly sharing it
* I looked at a Bible verse and considered sharing it or other options the app offered me.

Application Transition:





Selected Task Label:

I'm reading the terms of use agreement.

Resources Used:

phone number, and unique device identifier

Crowd expected resource use:

No

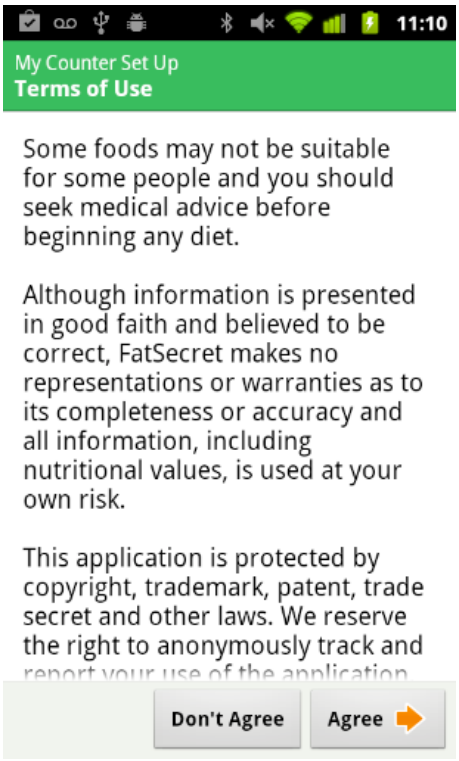
Crowd comfort [-2,2] (N, (μ,σ)):

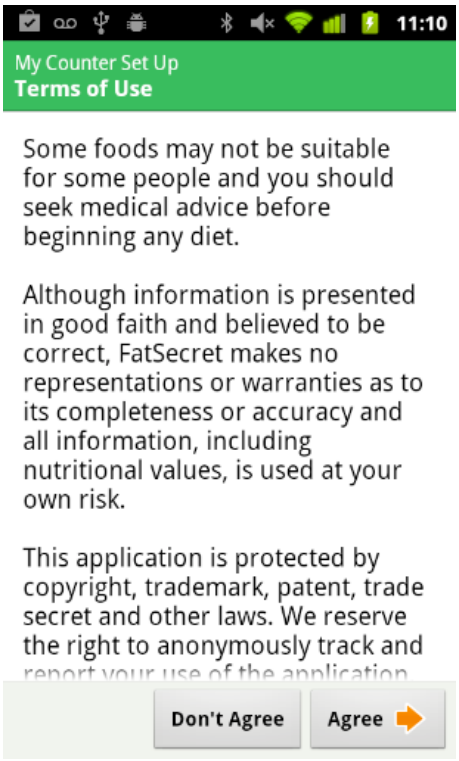
N=10, (μ=-0.9, σ=1.3)

Crowd-sourced Task Labels:

* Agreeing to the terms of use.
* accepting the terms and condtions
* Reading TOS
* users term on FatSecret
* signing the EULA
* read terms of use
* read terms of use
* agreeing to the terms and services to start using the app.
* Read Terms and Conditions
* I'm reading the terms of use agreement.

Application Transition:





Selected Task Label:

I read the terms of use for the app, and now I'm setting it up.

Resources Used:

phone number, and unique device identifier

Crowd expected resource use:

Yes

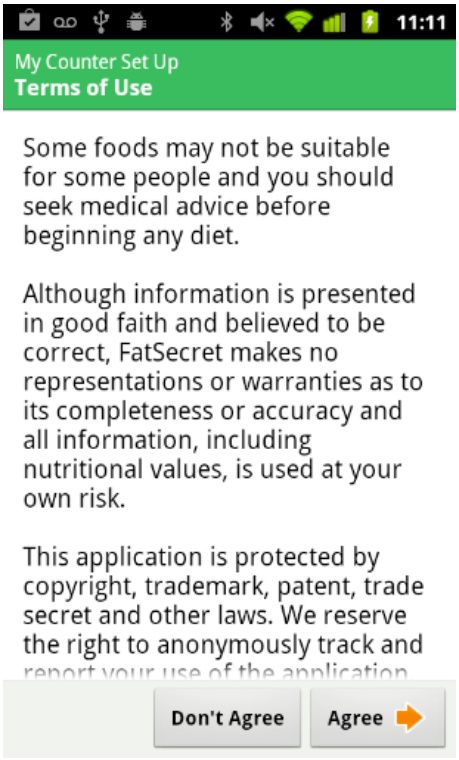
Crowd comfort [-2,2] (N, (μ,σ)):

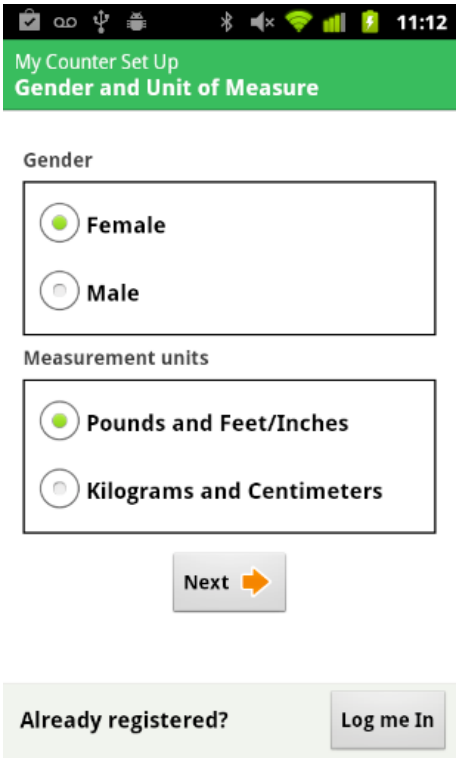
N=10, (μ=0.1, σ=1.1)

Crowd-sourced Task Labels:

* Putting in my measurements.
* Agreeing to TOS and starting app
* setting up profile for app
* Reading the disclosure then filling in my demographics
* Setting up your measurement
* entering in my specific information to form a plan to myself and set up my own acocunt
* read terms of use, set up account
* set up gender and unit of measurement
* Setup Application
* I read the terms of use for the app, and now I'm setting it up.

Application Transition:





Selected Task Label:

Checking work out statistics.

Resources Used:

exact location, and approximate location

Crowd expected resource use:

Yes

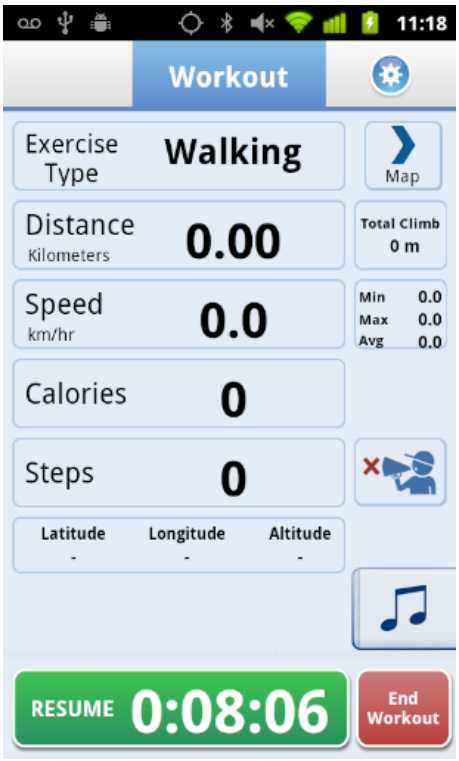
Crowd comfort [-2,2] (N, (μ,σ)):

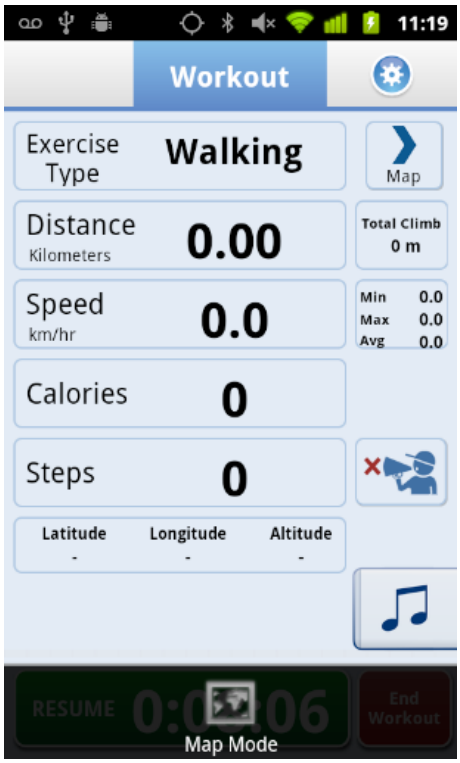
N=10, (μ=1.1, σ=1.4)

Crowd-sourced Task Labels:

* Track workout
* Mapping the 8 minute workout.
* Looking at statistics about my workout.
* checking your status
* Looks like I was trying to start a walking workout that will log my steps and possibly look at the map of where i will be walking.
* tracking a workout
* Hitting settings and bringing up map mode
* Checking work out statistics.
* Tracking a walk
* Resuming the walking workout and opening map mode.

Application Transition:





Selected Task Label:

track distance and time during exercise

Resources Used:

exact location, and approximate location

Crowd expected resource use:

Yes

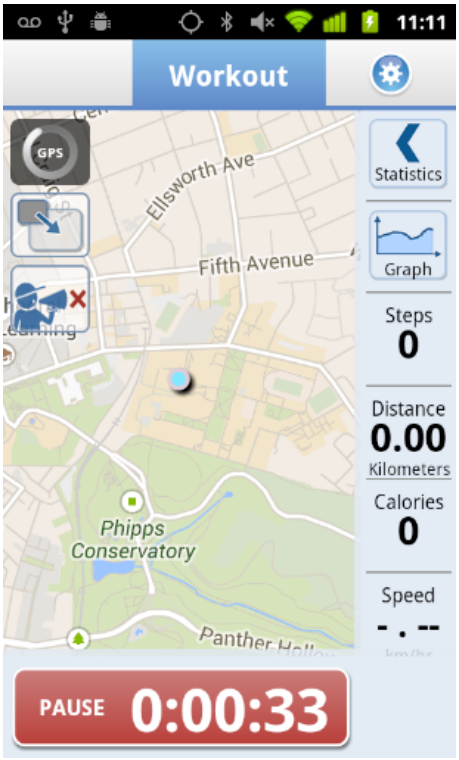
Crowd comfort [-2,2] (N, (μ,σ)):

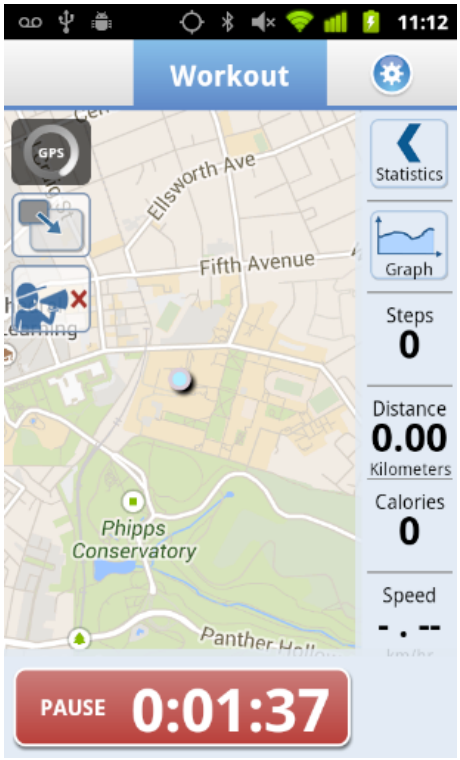
N=10, (μ=1.5, σ=0.9)

Crowd-sourced Task Labels:

* Measuring the duration and distance of my workout.
* Tracking and timing your workout
* checking your route
* GPS and track workout
* Starting a workout route.
* see your current location and workout stats
* Checking your location
* Tracking run
* Pausing the current workout on the app.
* track distance and time during exercise

Application Transition:





Selected Task Label:

Reading an article and then loading a video.

Resources Used:

approximate location

Crowd expected resource use:

No

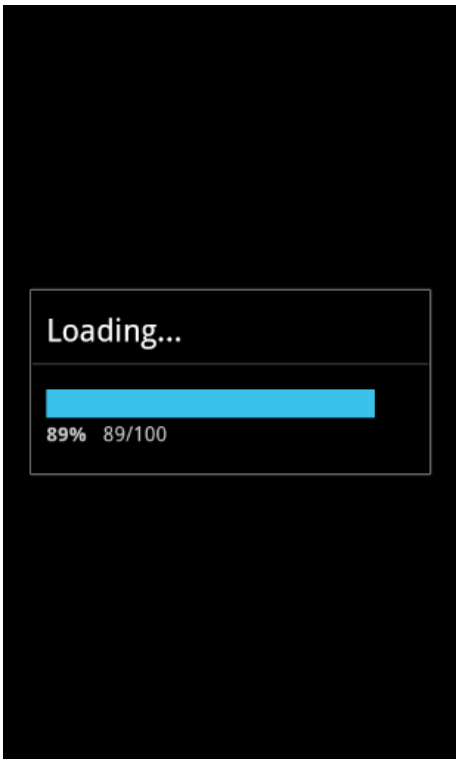
Crowd comfort [-2,2] (N, (μ,σ)):

N=10, (μ=-0.7, σ=1.4)

Crowd-sourced Task Labels:

* I selected a news story featured in the app and now I'm waiting for it to load.
* Load a news story
* going from an article to loading another
* Going to be previous screen
* keep updated with whats going on around the world. Download a video
* Loading page
* loading up the news
* Reading an article and then loading a video.
* view a story, view the picture

Application Transition:



Selected Task Label:

going back to the news stories

Resources Used:

approximate location

Crowd expected resource use:

No

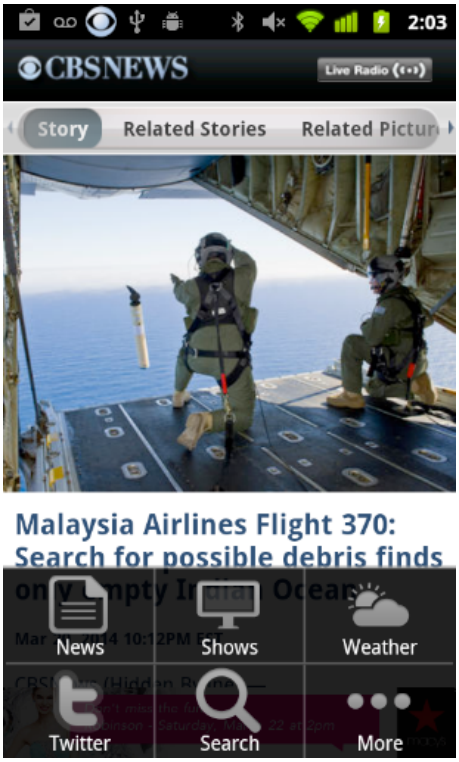
Crowd comfort [-2,2] (N, (μ,σ)):

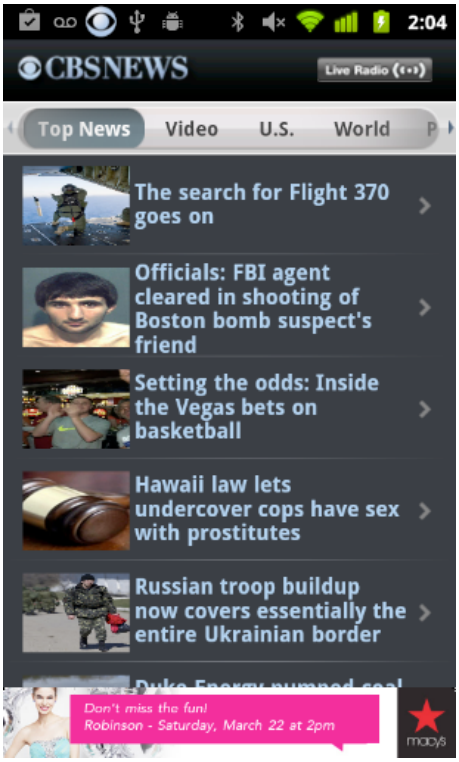
N=10, (μ=-0.2, σ=1.2)

Crowd-sourced Task Labels:

* Looking through videos
* Read the previous news article and went back to read most stories
* keep updated with whats going on around the world. and share a story
* looking at an article and looking at menu
* I finished with one news story then went back to the menu to browse others.
* Reading a story and then going to the current headlines.
* reading the latest news
* going back to the news stories
* viewing a story, then going back to top news
* From the menu screen going directly to viewing the top news.

Application Transition:





Selected Task Label:

I'm leaving feedback about the app.

Resources Used:

phone number, exact location, and approximate location

Crowd expected resource use:

No

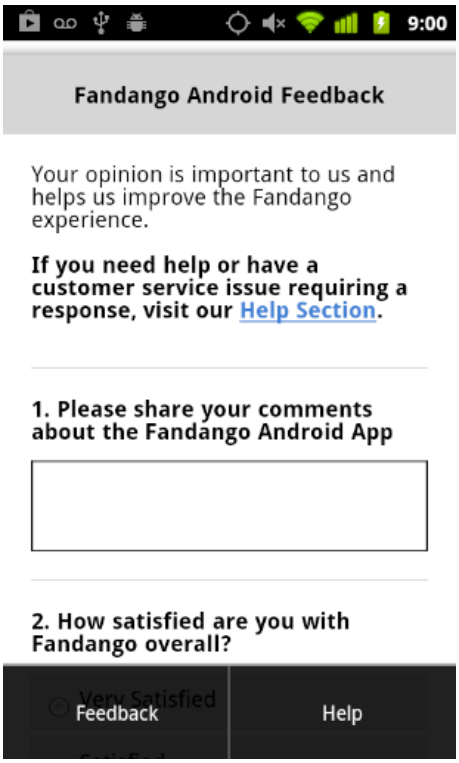
Crowd comfort [-2,2] (N, (μ,σ)):

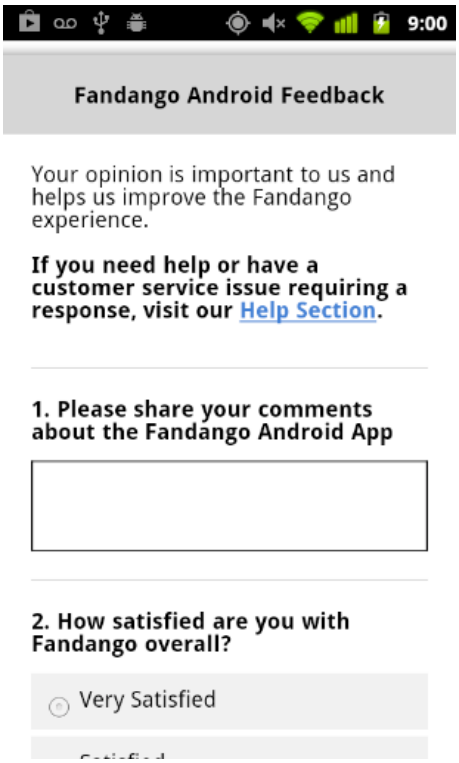
N=10, (μ=-0.5, σ=1.3)

Crowd-sourced Task Labels:

* watch a movie
* feedback on app
* I am providing feedback on the Fanango app
* give feedback on app
* Fandango Feedback
* I'm leaving feedback about the app.
* providing Fandango feedback
* give feedback
* searching for a movie playing in my area
* either trying to access help for the app or leaving feedback about the app

Application Transition:





Selected Task Label:

looking for dining experiences

Resources Used:

phone number, exact location, and approximate location

Crowd expected resource use:

Yes

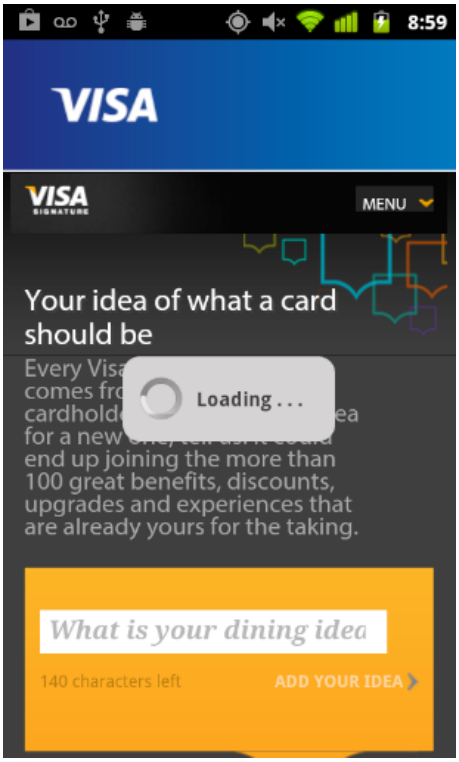
Crowd comfort [-2,2] (N, (μ,σ)):

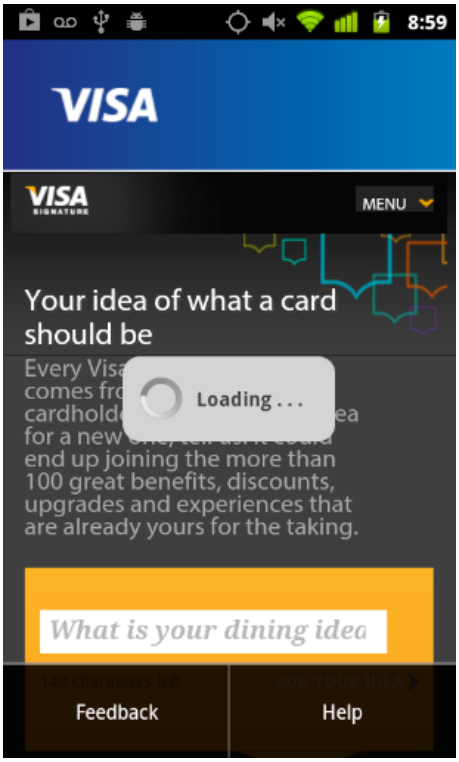
N=10, (μ=1.4, σ=0.5)

Crowd-sourced Task Labels:

* adding credit card to app
* Look for a restaurant.
* I had trouble with a page loading so I'm trying to get help online or leave feedback about it.
* looking for dining experiences
* entering a Visa idea
* load a payment type
* Perks associated with using your Visa
* Get help or feedback
* I am trying to look up for a place to eat that I can use my visa benefits at.
* searching for a restaurant to redeem Visa points at

Application Transition:





Selected Task Label:

turn on flashlight

Resources Used:

phone number, exact location, approximate location, and unique device identifier

Crowd expected resource use:

No

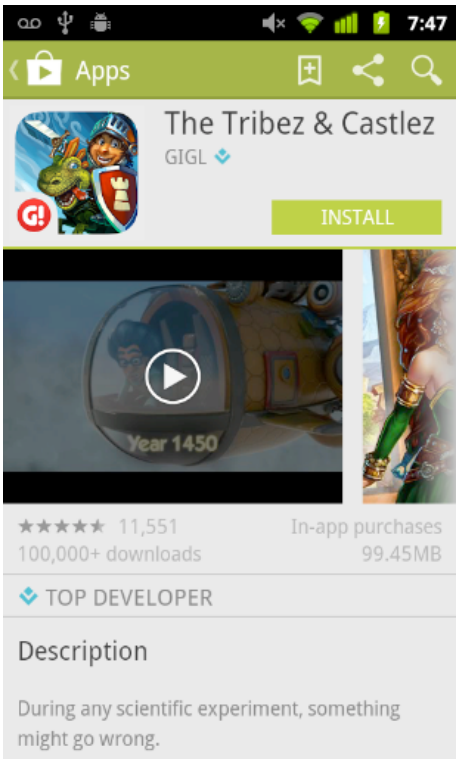
Crowd comfort [-2,2] (N, (μ,σ)):

N=10, (μ=-1.6, σ=0.5)

Crowd-sourced Task Labels:

* Install app
* installing the app
* turn the flashlight on
* play a game, find a cool light
* Download app, turn on flashlight
* Installing and launching the app.
* looking for a new app (and possibly installing the game) and then turning the flashlight on.
* Install app, turn on flashlight
* turn on flashlight

Application Transition:



Selected Task Label:

Rate it 5-Star

Resources Used:

phone number, exact location, approximate location, and unique device identifier

Crowd expected resource use:

No

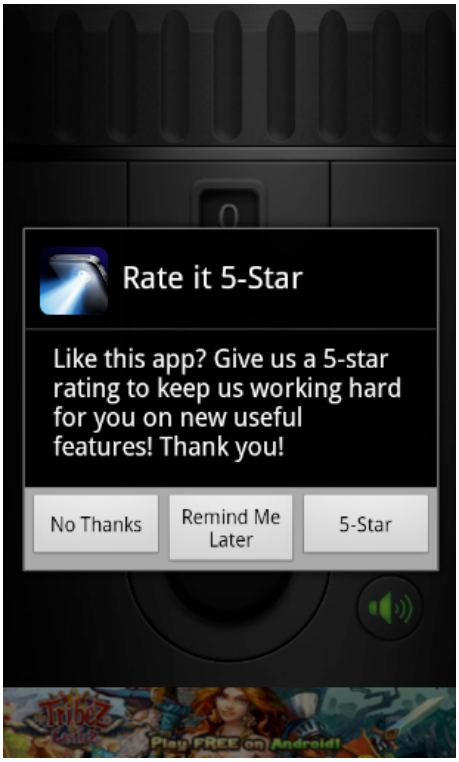
Crowd comfort [-2,2] (N, (μ,σ)):

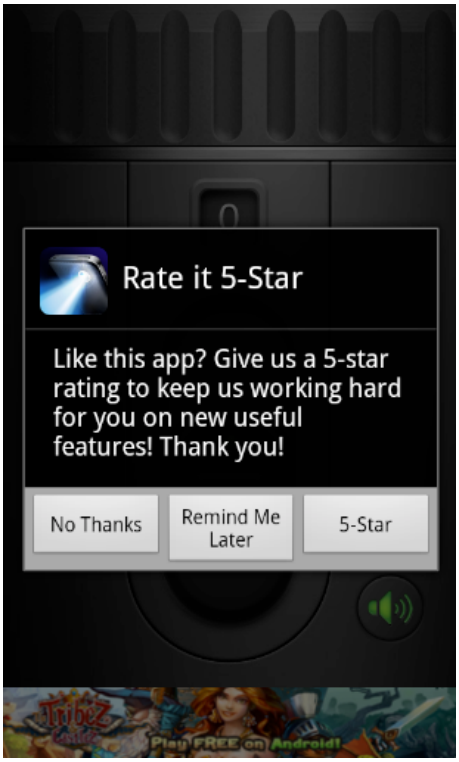
N=10, (μ=-0.9, σ=1.3)

Crowd-sourced Task Labels:

* Rate it 5-Star
* rate the app
* rating the app
* Just finished initial use of app and it wants me to rate it
* Respond to prompt
* was probably trying to turn the flashlight on when the rating popup popped up
* Respond to prompt
* Rate application
* It is trying to get the user to rate the app.
* use as a light game

Application Transition:





Selected Task Label:

Log into site, return to base login

Resources Used:

phone number, exact location, approximate location, and unique device identifier

Crowd expected resource use:

Yes

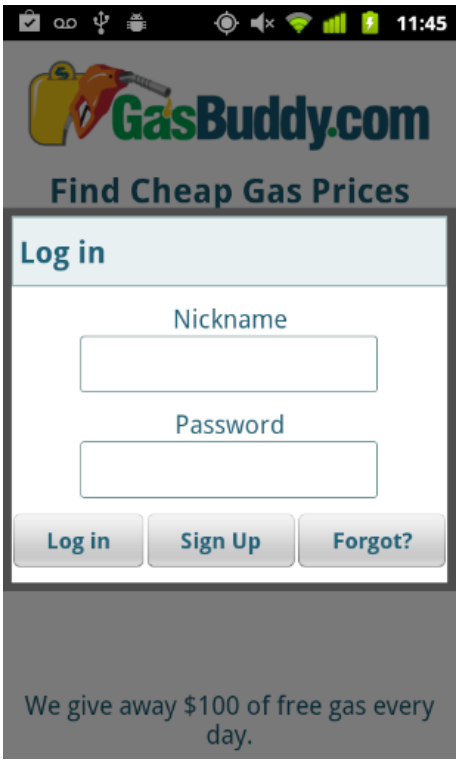
Crowd comfort [-2,2] (N, (μ,σ)):

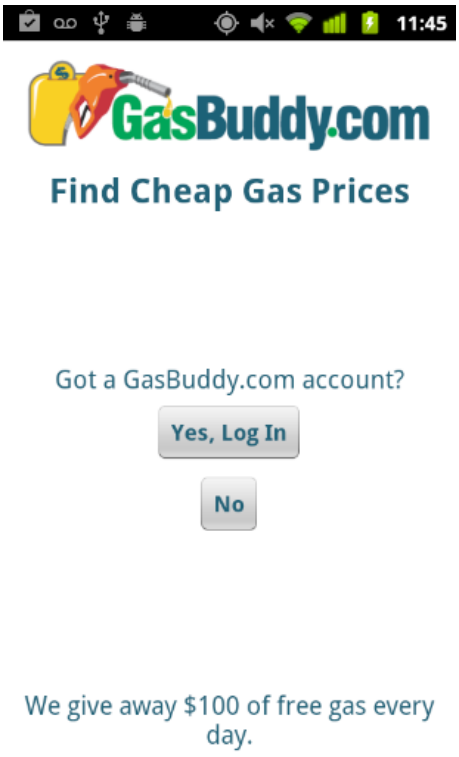
N=10, (μ=-0.2, σ=1.2)

Crowd-sourced Task Labels:

* log in
* logging in your account
* Log in, ask for login
* Signing into the app.
* failing to log in and going back to the option of logging in or signing up
* log in into chap gas app
* pressed the wong thing and went back to create an account
* log in
* login, view gas buddy
* Log into site, return to base login

Application Transition:





Selected Task Label:

find gas close to where I am

Resources Used:

phone number, exact location, and approximate location

Crowd expected resource use:

Yes

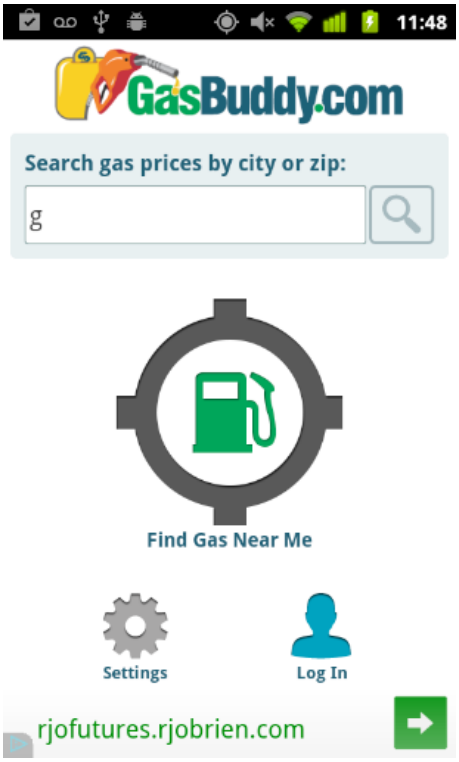
Crowd comfort [-2,2] (N, (μ,σ)):

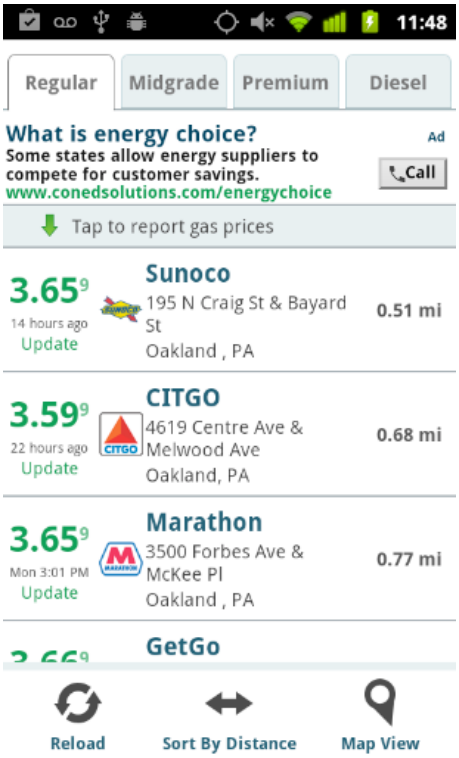
N=10, (μ=1.2, σ=0.9)

Crowd-sourced Task Labels:

* Search for gas, browse through stations
* checking out the best gas price
* find cheap gas
* searching for gas prices by location
* Searching gas prices.
* I am looking for cheap gas near me
* Find gas near me
* find gas close to where I am
* searched for cheap gas
* Search for gas, browse through stations

Application Transition:





Selected Task Label:

I would be trying to sign in using my facebook account, or trying to link my groupon and facebook accounts together.

Resources Used:

phone number, exact location, and approximate location

Crowd expected resource use:

No

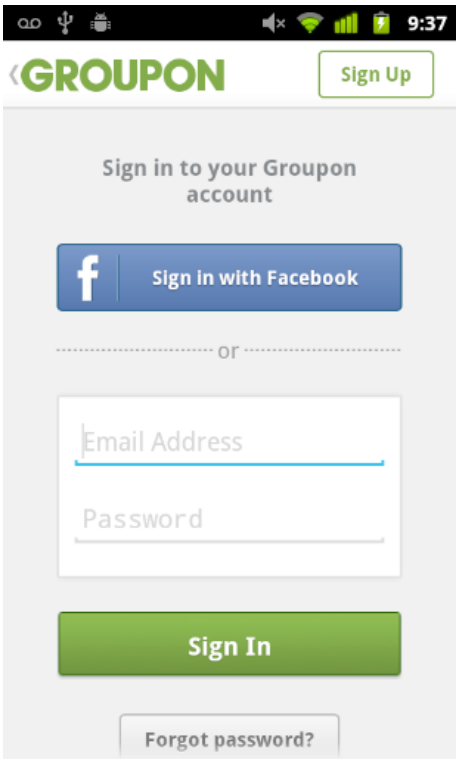
Crowd comfort [-2,2] (N, (μ,σ)):

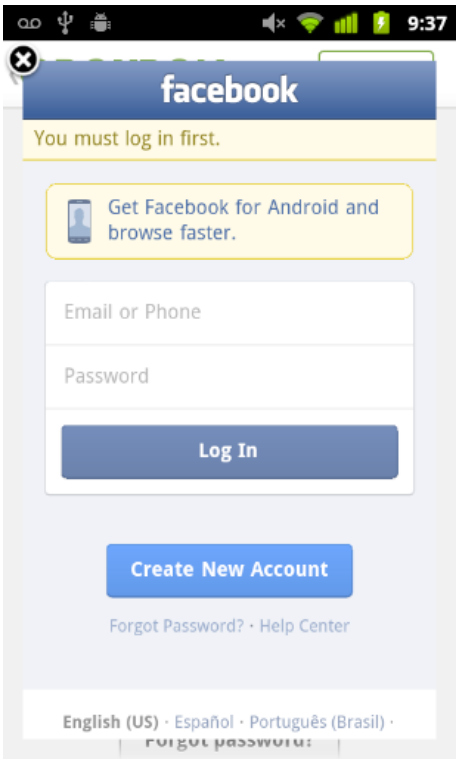
N=9, (μ=0.0, σ=1.2)

Crowd-sourced Task Labels:

* trying to log in with facebook but having to log into facebook first
* Log in using Facebook
* sign in with facebook
* Sign up, login with FB
* logging into the app
* signing into facebook
* Using my facebook account login to sign up for a groupon account
* I would be trying to sign in using my facebook account, or trying to link my groupon and facebook accounts together.
* signing into groupon through facebook
* Sign in

Application Transition:





Selected Task Label:

Search for groupons for a specific location

Resources Used:

phone number, exact location, and approximate location

Crowd expected resource use:

Yes

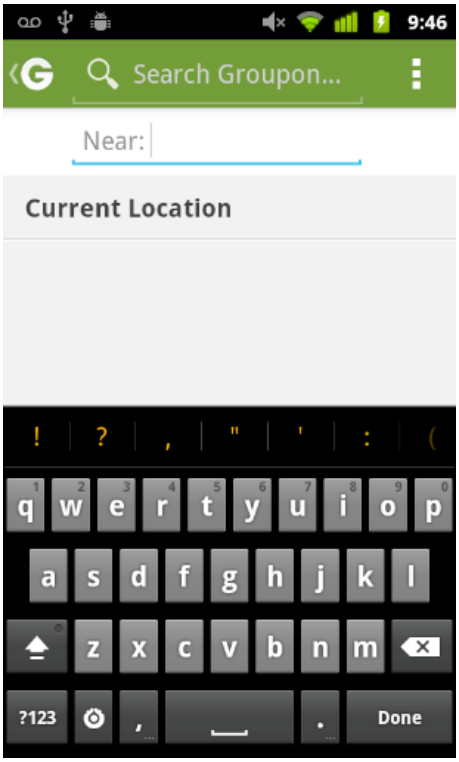
Crowd comfort [-2,2] (N, (μ,σ)):

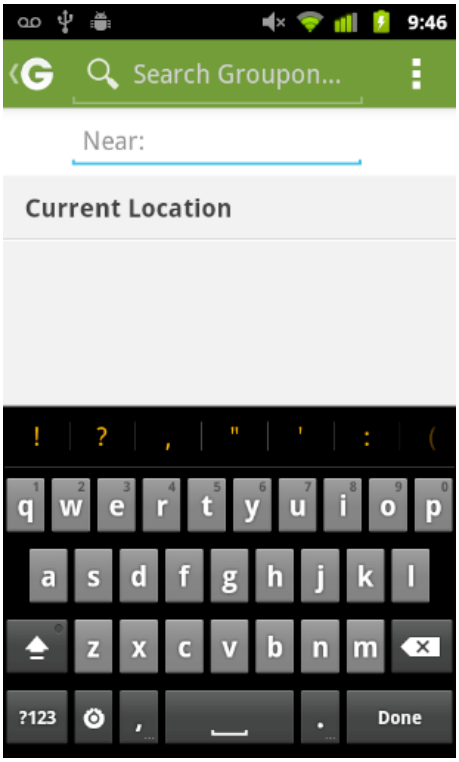
N=10, (μ=0.8, σ=1.5)

Crowd-sourced Task Labels:

* Search for groupons for a specific location
* Trying to find current groupon deals near my current location
* find current location
* Enter location
* find deals near a location
* searching nearby
* Search for Groupon
* entering location into the groupon app
* Search for a coupon.
* Find a deal close to me.

Application Transition:





Selected Task Label:

check my horoscope

Resources Used:

phone number, exact location, approximate location, and unique device identifier

Crowd expected resource use:

No

Crowd comfort [-2,2] (N, (μ,σ)):

N=10, (μ=-1.5, σ=0.9)

Crowd-sourced Task Labels:

* check my horoscope
* checking out your life in the future
* Find out astrological predictions for the Aries sign for today.
* See what Aries says
* selecting your sign
* Get more info about aries.
* I can click on my specific horoscope
* Read my horoscope
* Daily reading

Application Transition:

Selected Task Label:

Looking at a horoscope then opening a menu with sharing and email options for it.

Resources Used:

phone number, exact location, approximate location, and unique device identifier

Crowd expected resource use:

No

Crowd comfort [-2,2] (N, (μ,σ)):

N=9, (μ=-0.9, σ=1.1)

Crowd-sourced Task Labels:

* checking out your life in the future
* share a link
* Select the settings menu
* Looking at a horoscope then opening a menu with sharing and email options for it.
* sharing a link
* Share my horoscope on a social networking site.
* Look for more options
* share horoscope
* Share info with friends.
* share my horoscope or contact the app developers

Application Transition:

Selected Task Label:

Starting the app and reading the legal notice

Resources Used:

exact location, and approximate location

Crowd expected resource use:

Yes

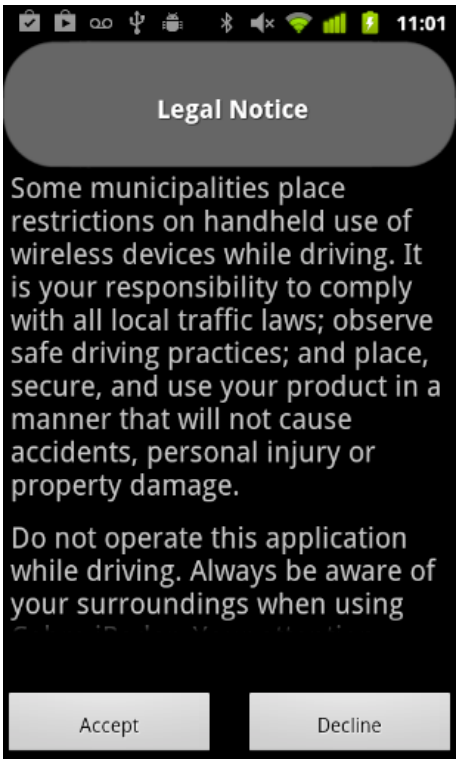
Crowd comfort [-2,2] (N, (μ,σ)):

N=10, (μ=0.5, σ=1.0)

Crowd-sourced Task Labels:

* accepting the terms and condtions
* Tells you where traffic cameras are.
* I am accepting or declining the legal notice
* turn on voice features for driving
* Opening the app and signing my consent to use it
* Launching Application
* starting app for the first time
* Starting the app and reading the legal notice
* Find app, accept legal notice
* Find app, accept notice

Application Transition:



Selected Task Label:

Running a report for Radar detectors

Resources Used:

exact location, and approximate location

Crowd expected resource use:

Yes

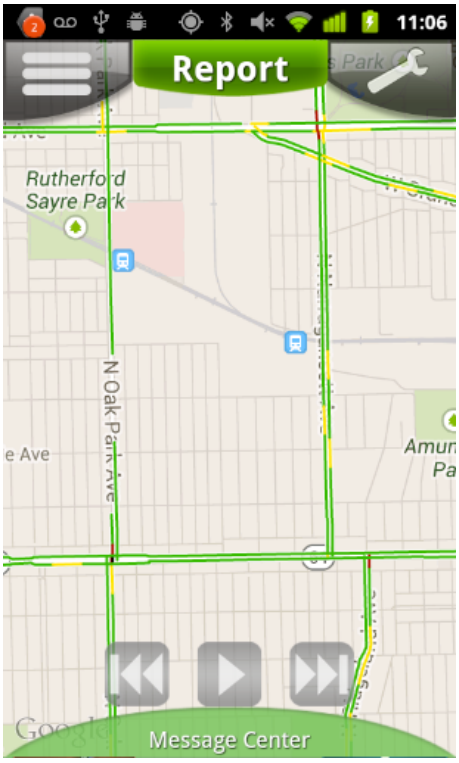
Crowd comfort [-2,2] (N, (μ,σ)):

N=10, (μ=0.8, σ=1.0)

Crowd-sourced Task Labels:

* result is all clear
* I guess it would tell you where your friends are while they're calling you.
* I am checking for radars within a certain area.
* searching for my house
* finding the direction
* Trying to allocate whether any radar or laser is active in my area
* home screen to map
* Report speed trap
* Running a report for Radar detectors
* Find app, use map

Application Transition:



Selected Task Label:

It looks like I was trying to save something or go into the settings.

Resources Used:

approximate location

Crowd expected resource use:

No

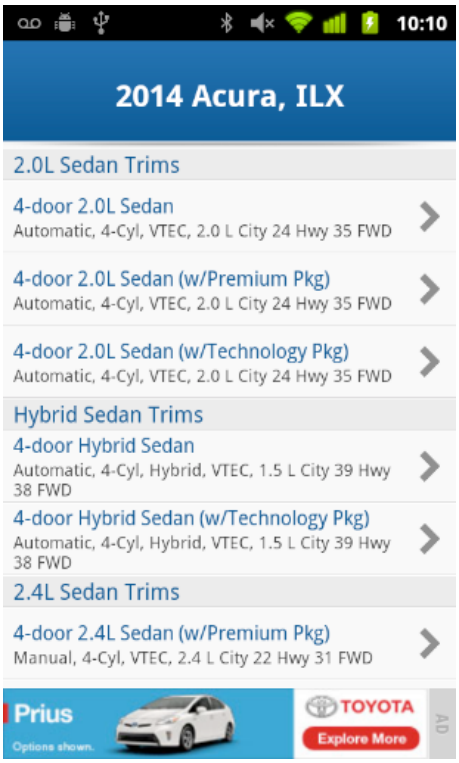
Crowd comfort [-2,2] (N, (μ,σ)):

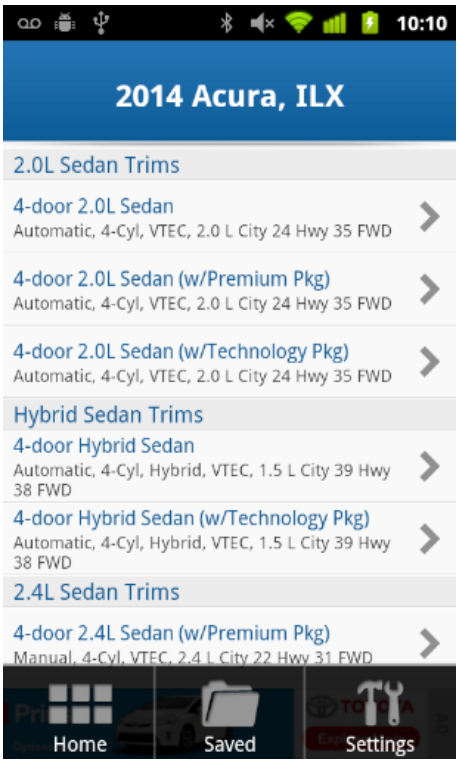
N=10, (μ=-0.2, σ=1.2)

Crowd-sourced Task Labels:

* browsing the list of cars on sale
* search through 2014 acura, ilx
* Finding prices on vehicle packages for the 2014 Acura ILX.
* Find an Acura with the features of my preference.
* Browse 2014 Acura ILX
* Search for a specific model of a 2014 Acura ILX
* Bringing up a toolbar at the bottom to navigate away to another function.
* It looks like I was trying to save something or go into the settings.
* Looking at 2014, Acura ILX and its configurations and then going back into settings.
* Look for listing

Application Transition:





Selected Task Label:

KBB App homepage

Resources Used:

approximate location

Crowd expected resource use:

Yes

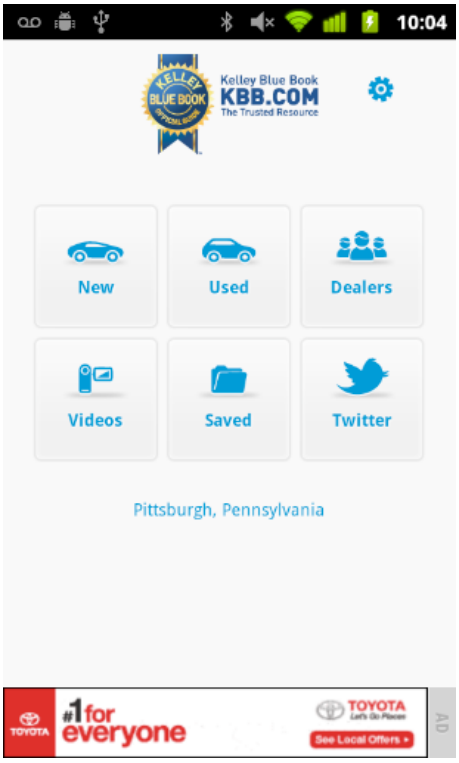
Crowd comfort [-2,2] (N, (μ,σ)):

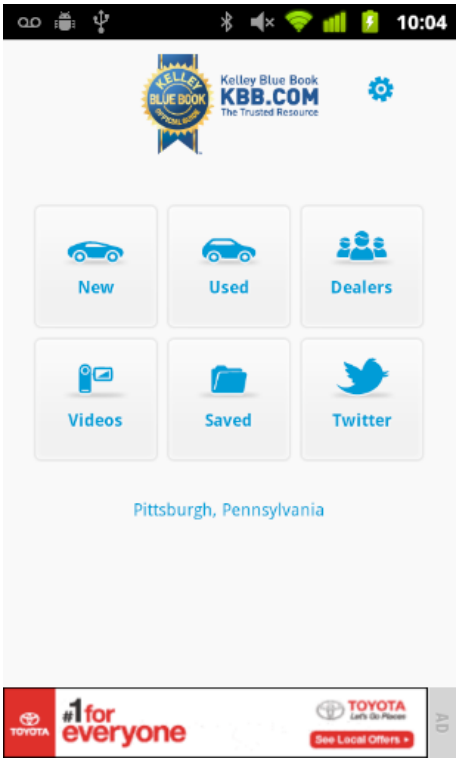
N=10, (μ=1.8, σ=0.4)

Crowd-sourced Task Labels:

* begin using the app
* Looking for Kelley Blue Book prices on cars.
* Looking at the front section of the kellyes blue book.
* you're about to start using the app
* KBB App homepage
* search for a car
* I have just opened the app and am preparing to navigate to something within it.
* Choosing what kind of car you're looking for
* Navigate through the KBB.com app.
* see the options on the home screen

Application Transition:





Selected Task Label:

Bringing up a menu with additional options from the map screen.

Resources Used:

exact location, and approximate location

Crowd expected resource use:

Yes

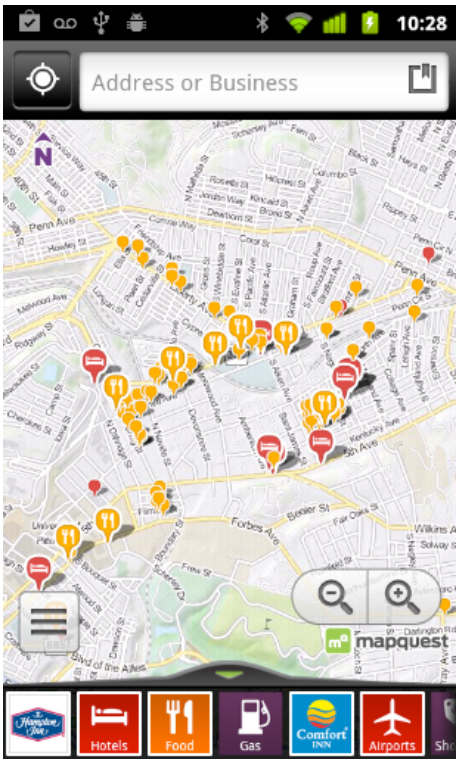
Crowd comfort [-2,2] (N, (μ,σ)):

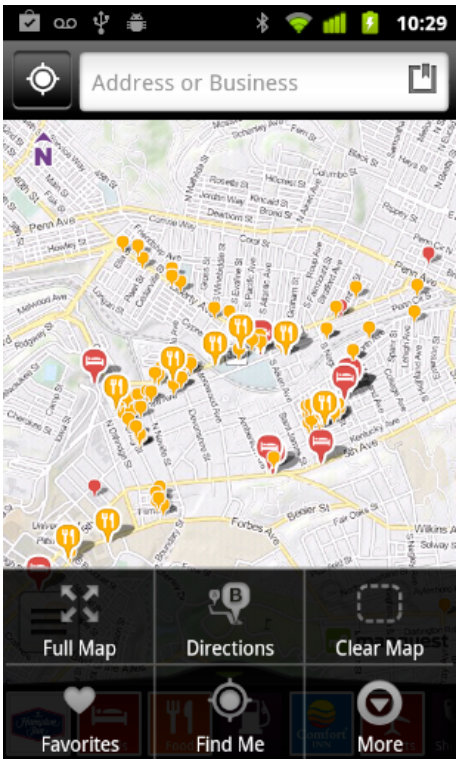
N=10, (μ=1.0, σ=1.1)

Crowd-sourced Task Labels:

* Finding locations of certain places
* Get directions to a business.
* I was looking at the map and then went to the settings.
* find certain locations
* Looking at pinpoints of businesses around me, then opening the menu.
* finding friends and places to eat/gas/hotel
* I am looking at the map with hotels and food
* Opening an options menu
* Adding local Hotels, Restaurants and attractions on the map.
* Bringing up a menu with additional options from the map screen.

Application Transition:





Selected Task Label:

Show location on GPS

Resources Used:

exact location, and approximate location

Crowd expected resource use:

Yes

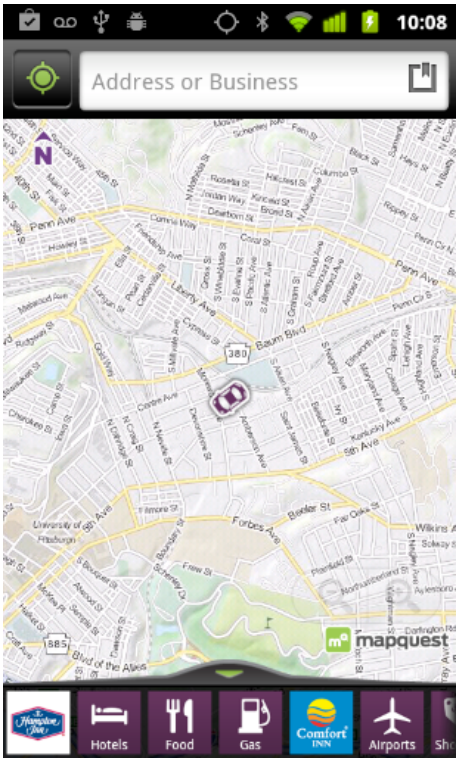
Crowd comfort [-2,2] (N, (μ,σ)):

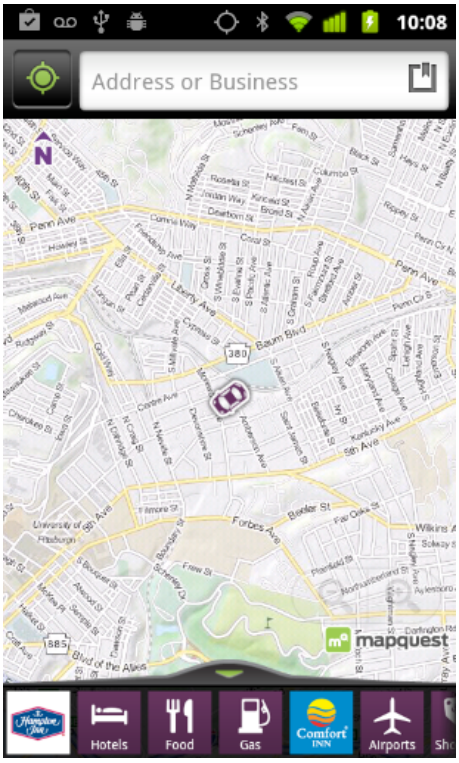
N=10, (μ=1.7, σ=0.5)

Crowd-sourced Task Labels:

* Looking at map; searching for nearby business
* Map my current location.
* A map where you search for places
* I am getting navigational directions while driving
* finding out where you are via gps
* Looking for a rest area nearby
* Finding where I am on the map
* Show location on GPS
* With GPS enabled looking for Inn's and Hotels on the map.
* Looking for strictly inns and hotels using your location.

Application Transition:





Selected Task Label:

Go back to the menu before the OS version.

Resources Used:

approximate location, and unique device identifier

Crowd expected resource use:

No

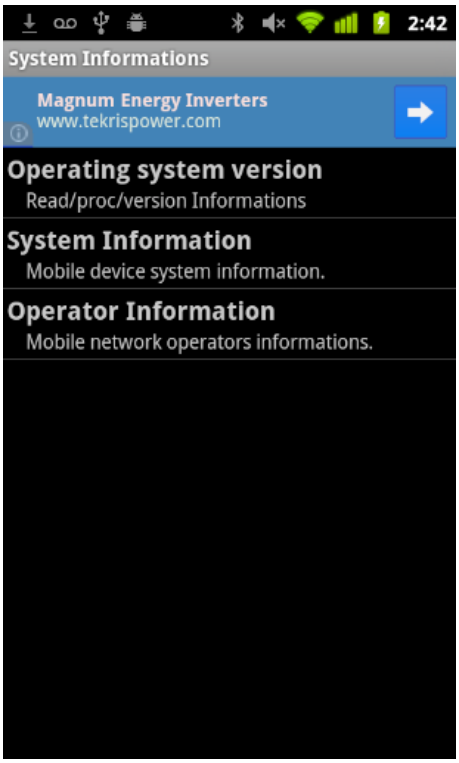
Crowd comfort [-2,2] (N, (μ,σ)):

N=10, (μ=-1.2, σ=0.9)

Crowd-sourced Task Labels:

* Go back to info screen
* Finding information about my phone's operating system.
* looking at system info on phone
* obtain info on operating system
* Go back to system information
* Going from Operating sytem version screen to the system info screen.
* looking at operating version, then system information
* Went back to the pervious screen with the different information types
* Go back to the menu before the OS version.
* Find system information

Application Transition:



Selected Task Label:

I pulled up my operating system information and I can share it with someone via bluetooth or messaging.

Resources Used:

approximate location, and unique device identifier

Crowd expected resource use:

Yes

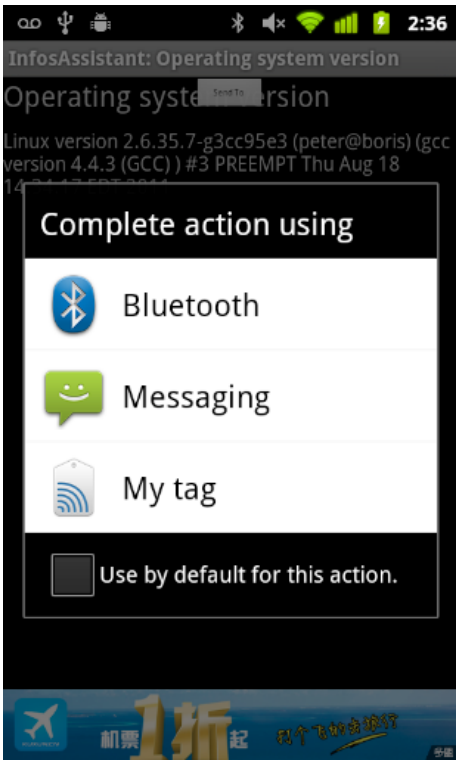
Crowd comfort [-2,2] (N, (μ,σ)):

N=10, (μ=0.7, σ=1.4)

Crowd-sourced Task Labels:

* See the operating system version
* Picking which device to send information to
* looking at system info on phone and clicking an ad
* Sending my phone's operating system information via bluetooth or messaging.
* To see what version of operating system
* sending my operating system info somewhere
* Select action method
* I would be testing out new messaging software or blue-tooth drivers/software.
* I pulled up my operating system information and I can share it with someone via bluetooth or messaging.
* Looking at running programs and which operating system I am using

Application Transition:



Selected Task Label:

Sign in With Facebook.

Resources Used:

phone number, exact location, and approximate location

Crowd expected resource use:

No

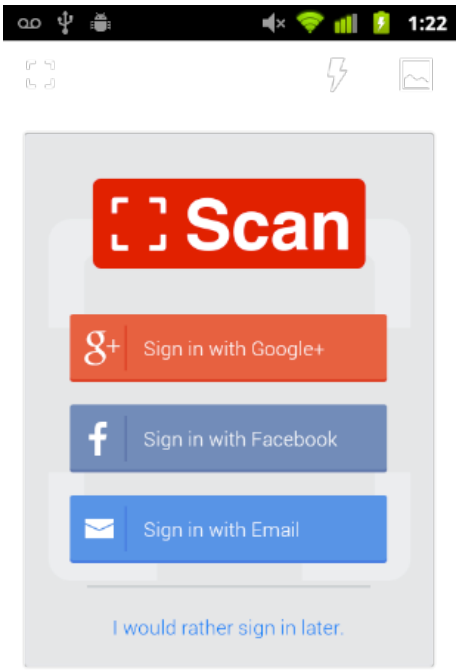
Crowd comfort [-2,2] (N, (μ,σ)):

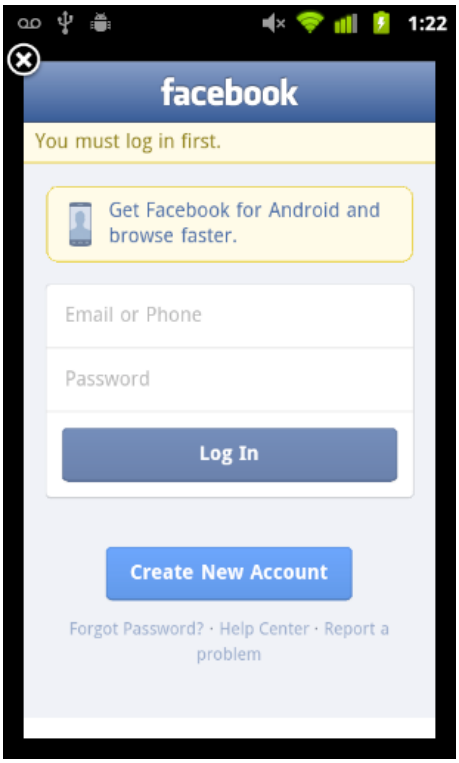
N=10, (μ=-0.8, σ=1.0)

Crowd-sourced Task Labels:

* signing up the app
* Sign in With Facebook.
* Select option, sign into FB
* Log in via Faceook from login screen
* Select, log into FB
* Connect with Facebook
* Scan a QR code
* trying to login to the app
* Scanned a code now sharing options to social networks. Picked facebook.
* Sign in with facebook

Application Transition:





Selected Task Label:

Log into FB

Resources Used:

phone number, exact location, and approximate location

Crowd expected resource use:

No

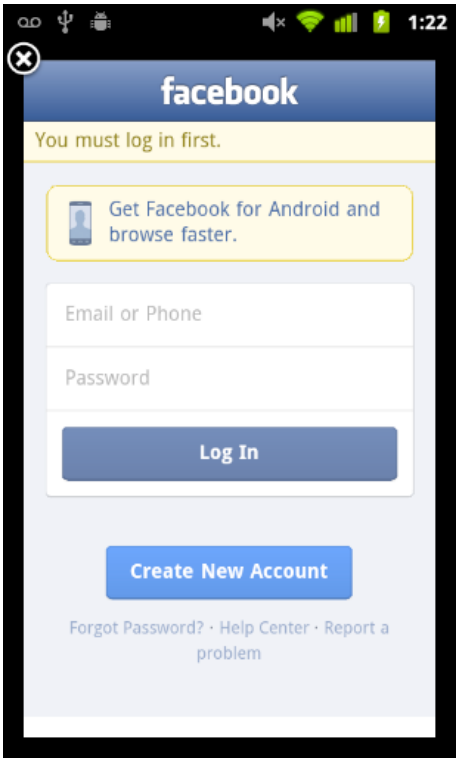
Crowd comfort [-2,2] (N, (μ,σ)):

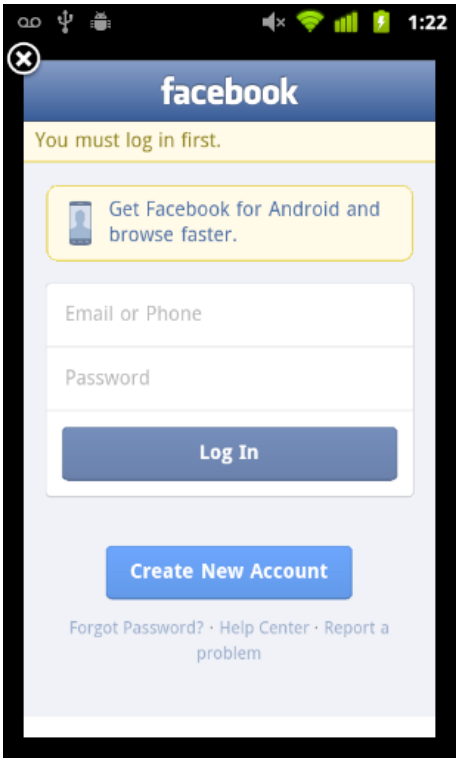
N=10, (μ=-0.9, σ=1.0)

Crowd-sourced Task Labels:

* signing up the app through facebook
* Registering via facebook.
* Log into facebook.
* Log in via Facebook
* Log into FB
* Log into FB
* Log into facebook.
* trying to login to the app
* Trying to login into facebook
* Want to make your own QR codes for others to scan

Application Transition:





Selected Task Label:

sharing a link

Resources Used:

phone number, exact location, and approximate location

Crowd expected resource use:

No

Crowd comfort [-2,2] (N, (μ,σ)):

N=9, (μ=-1.2, σ=0.4)

Crowd-sourced Task Labels:

* Connecting to Facebook
* signing into facebook
* signing up on Shazam through facebook
* Share my Shazam activity on Facebook
* Trying to share tags with Facebook
* login to facebook to share with friends
* share with friends
* I am getting ready to sign into facebook
* Allowing shazam to post on facebook timeline
* sharing a link

Application Transition:

Selected Task Label:

Tried to shazam a song or video but the database could not find due to another app using the mic

Resources Used:

phone number, exact location, and approximate location

Crowd expected resource use:

No

Crowd comfort [-2,2] (N, (μ,σ)):

N=9, (μ=-1.2, σ=0.4)

Crowd-sourced Task Labels:

* Trying to see what song is playing
* Touch to Shazam
* getting an error using Shazam
* tag music
* tagging songs
* Tried to shazam a song or video but the database could not find due to another app using the mic
* identify music
* I received an error message for a tag I was creating
* Cannot tag because another app is using the mic
* To find out which song is playing

Application Transition:

Selected Task Label:

I am trying to leave feedback

Resources Used:

approximate location, and unique device identifier

Crowd expected resource use:

No

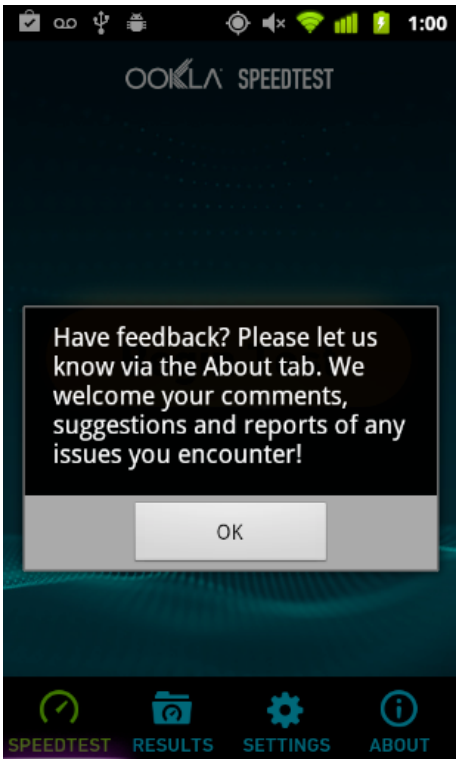
Crowd comfort [-2,2] (N, (μ,σ)):

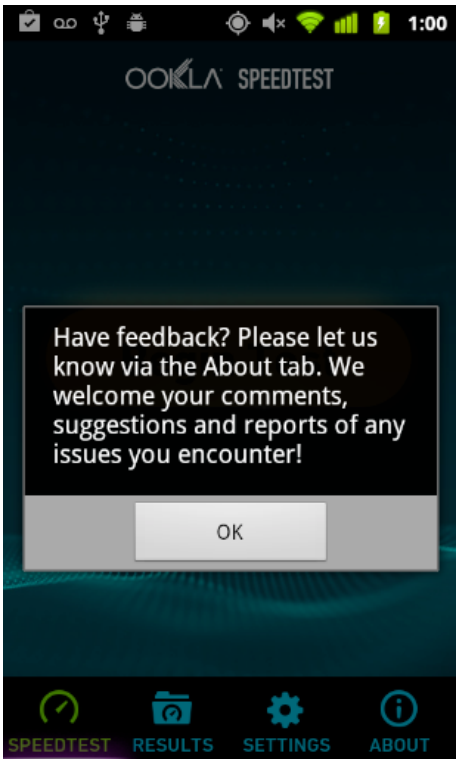
N=10, (μ=-0.3, σ=1.6)

Crowd-sourced Task Labels:

* I am trying to leave feedback
* leave feedback
* Leaving feedback about the app
* Feedback on the app
* Provide feedback to the app
* asking for feedback about the app
* starting app
* Answer prompt
* Respond to prompt
* Doing a speedtest and maybe leaving feedback.

Application Transition:





Selected Task Label:

Respond to prompt, begin test

Resources Used:

exact location, approximate location, and unique device identifier

Crowd expected resource use:

Yes

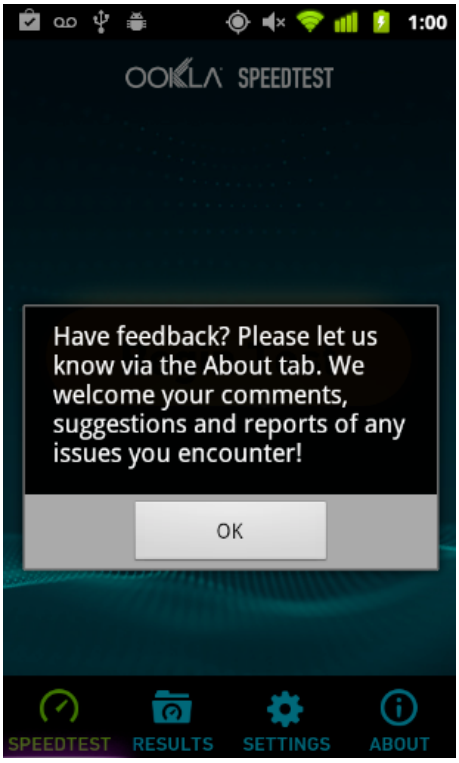
Crowd comfort [-2,2] (N, (μ,σ)):

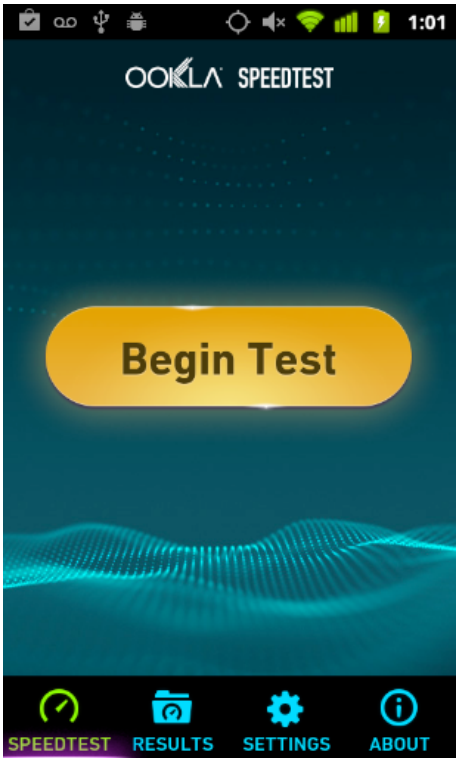
N=9, (μ=1.3, σ=0.9)

Crowd-sourced Task Labels:

* Leave feedback and start a speed test.
* start speed test
* Feedback on the app and Begin Test
* I just completed a speed test and am trying to take another one to verify the results
* Giving feedback information on App and about to begin speed test
* get past the opening screen to test the connection speed
* Test my internet speed.
* starting app to begin speed test
* Answer prompt
* Respond to prompt, begin test

Application Transition:





Selected Task Label:

Signing into Facebook instead of directly from the site.

Resources Used:

approximate location

Crowd expected resource use:

No

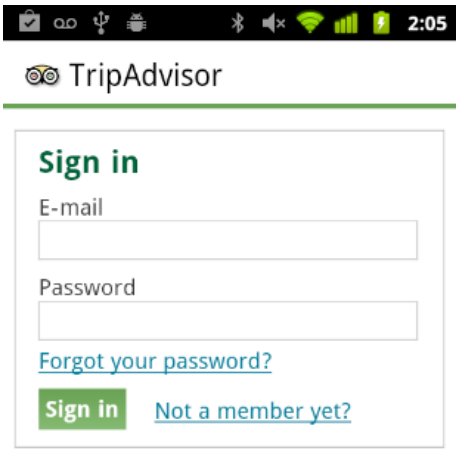
Crowd comfort [-2,2] (N, (μ,σ)):

N=10, (μ=0.0, σ=1.5)

Crowd-sourced Task Labels:

* signing in
* You went from a regular sign in page to a sign in with facebook page.
* Signing in to the app
* Log in to my TripAdvisor account.
* Signing into Facebook instead of directly from the site.
* log in using either my email address or with facebook
* log in TripAdvisor
* Returning to the app's title screen from the sign in screen without signing in.
* sign into app and open up app
* login to the app

Application Transition:



Selected Task Label:

look at other types of lodging types near me

Resources Used:

approximate location

Crowd expected resource use:

Yes

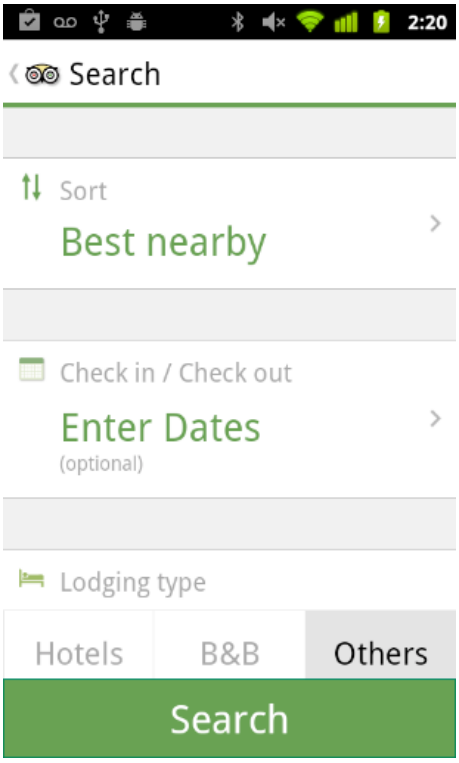
Crowd comfort [-2,2] (N, (μ,σ)):

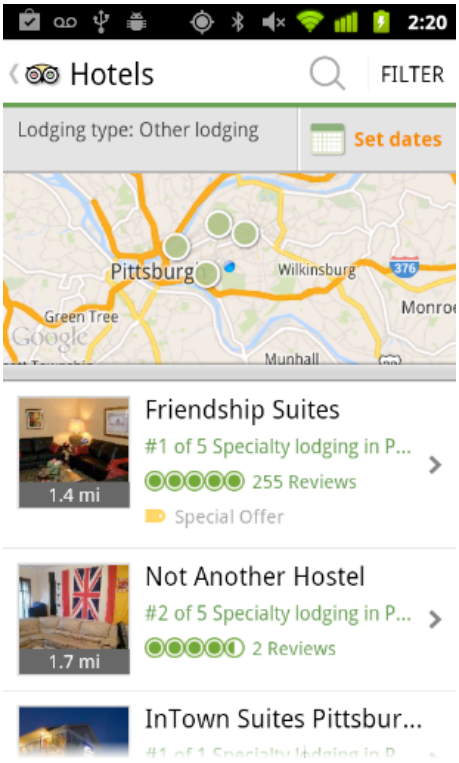
N=10, (μ=1.3, σ=1.2)

Crowd-sourced Task Labels:

* searching for hotels
* Search for hotels near Pittsburg.
* Searching for hotels nearbby
* Searching for the best nearby hostels and lodging by date.
* Looking for a hotel to stay at in the city of Pittsburg and the surrounding area.
* search for hotels
* select a date for hotel and look for a hotel nearby
* Using map to find nearby Hotels
* I looked at a list of hotels available for particular dates.
* look at other types of lodging types near me

Application Transition:





Selected Task Label:

accepting service terms and touring the app

Resources Used:

phone number, exact location, and approximate location

Crowd expected resource use:

No

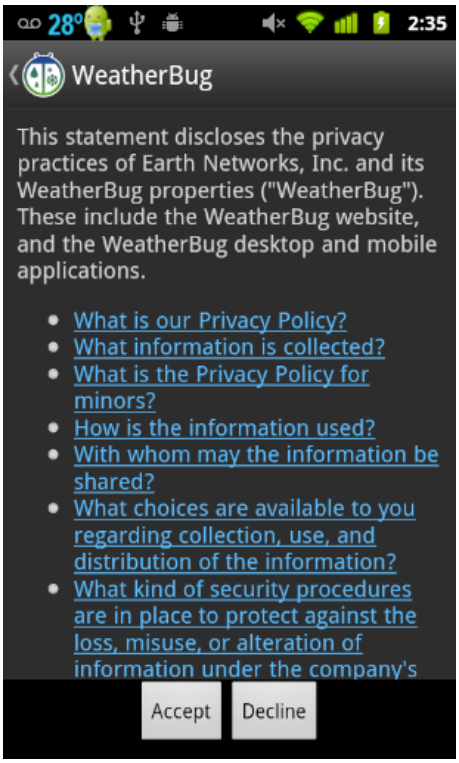
Crowd comfort [-2,2] (N, (μ,σ)):

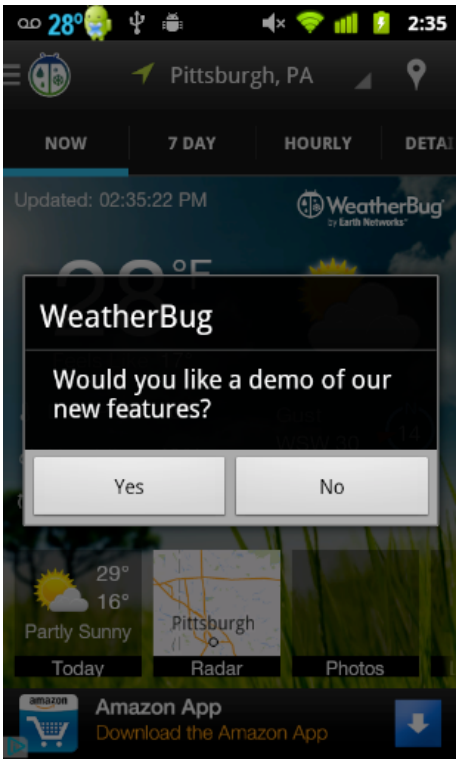
N=10, (μ=-0.1, σ=1.6)

Crowd-sourced Task Labels:

* asking for a demo about the app
* Looking at demo and privacy policy
* I accepted the terms for the app and I'm using it for the first time.
* Check to see what the app does with my information, and see what the app does.
* View Privacy information and preview a demo of new features
* Reading the Privacy Policy for WeatherBug and trying out a demo of their new features
* accepting service terms and touring the app
* reviewing privacy policy

Application Transition:





Selected Task Label:

refreshing the 7 day forecast

Resources Used:

phone number, exact location, and approximate location

Crowd expected resource use:

Yes

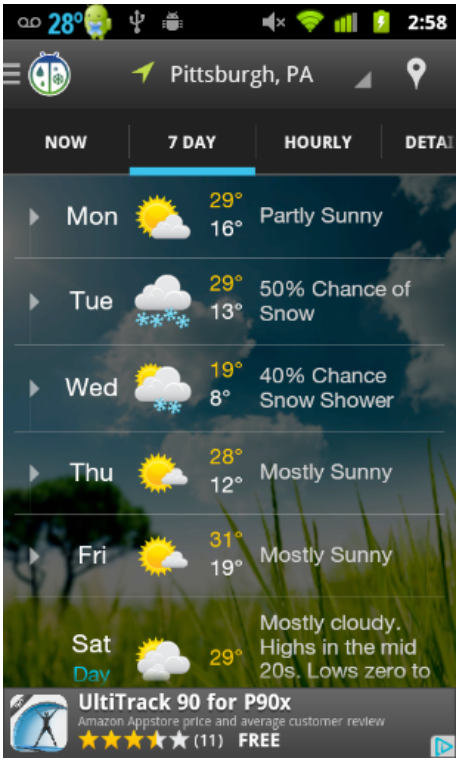
Crowd comfort [-2,2] (N, (μ,σ)):

N=10, (μ=1.1, σ=1.1)

Crowd-sourced Task Labels:

* checking forecast
* checking the weather for the whole week
* View weather in Pittsburgh and then refresh the screen
* Checking the 7 day forecast in my area
* Finding out the 7 day weather forecast for Pittsburgh, PA
* It looks like I was trying to see the forecast for the rest of the week
* refreshed the screen for the 7 day forecast
* refreshing the 7 day forecast
* 7-Day Forecast
* view weather

Application Transition:



Selected Task Label:

Using the tips

Resources Used:

approximate location, and unique device identifier

Crowd expected resource use:

No

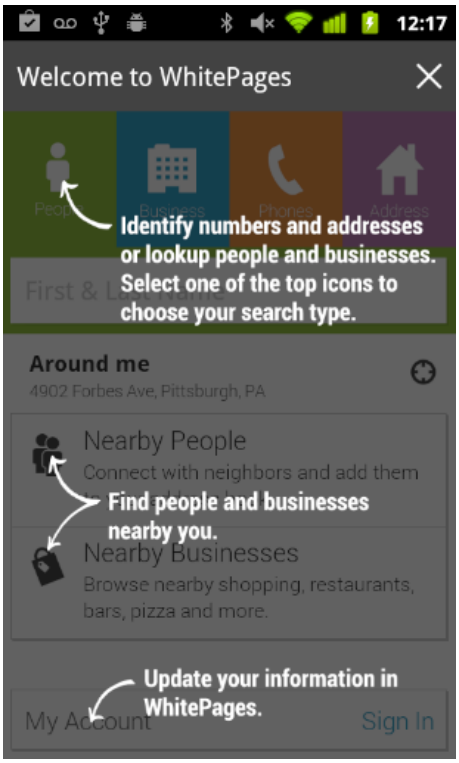
Crowd comfort [-2,2] (N, (μ,σ)):

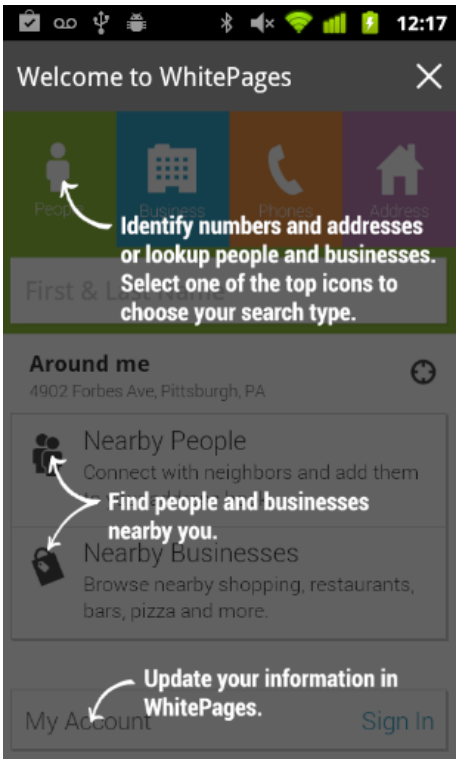
N=10, (μ=-0.4, σ=1.2)

Crowd-sourced Task Labels:

* Learn about White Pages
* Opening the app for the first time.
* Opening the app for the first time.
* introduction to application with tutorial.
* going through the app's tour
* Learning how to use the app
* an intro help screen to learn how to use the app
* finding people and business near you
* Going through tutorial
* Using the tips

Application Transition:





Selected Task Label:

starting at homepage, opening the application.

Resources Used:

approximate location

Crowd expected resource use:

Yes

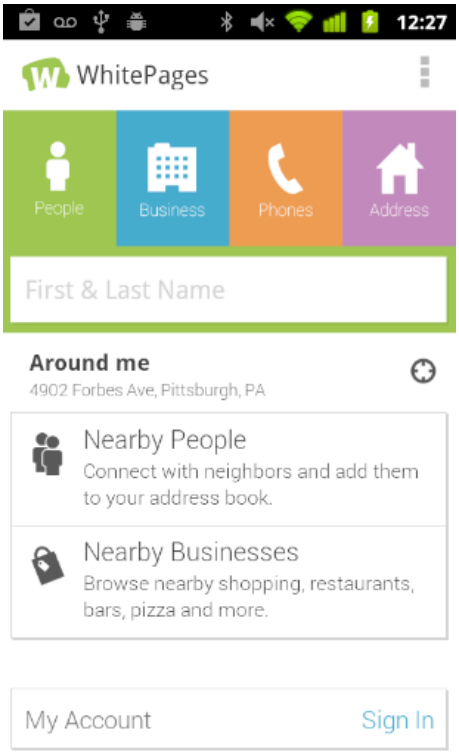
Crowd comfort [-2,2] (N, (μ,σ)):

N=10, (μ=1.0, σ=1.3)

Crowd-sourced Task Labels:

* Opening the home screen of the app
* Open White pages
* Starting the app and looking for things close to you
* Launch app.
* Search for a nearby address.
* starting at homepage, opening the application.
* Launch application
* make a call and look for business near by
* entering people's name to search

Application Transition:



Selected Task Label:

Find a business in Pittsburgh after being on your home screen on your phone.

Resources Used:

approximate location, and unique device identifier

Crowd expected resource use:

Yes

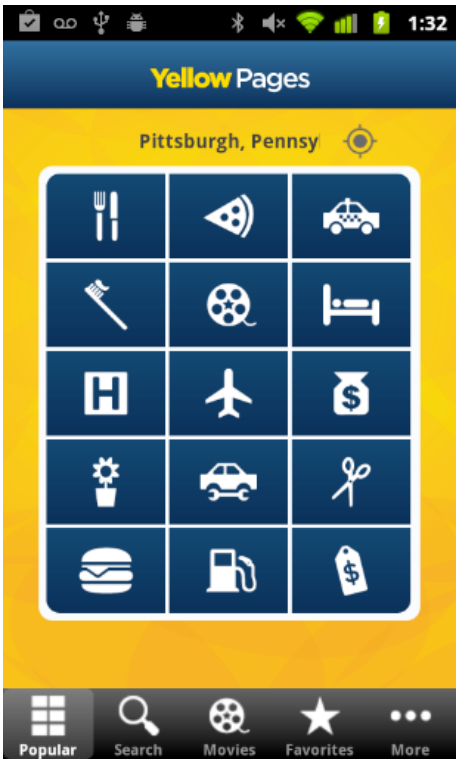
Crowd comfort [-2,2] (N, (μ,σ)):

N=10, (μ=0.6, σ=1.6)

Crowd-sourced Task Labels:

* starting the app
* opening the app
* Going from the home screen to the app
* Trying to search for something in the vicinity of the Pittsburgh metro area
* Opening the app to the home screen.
* Find a business in Pittsburgh after being on your home screen on your phone.
* From the homescreen opening the app and looking at the popular section.
* homepage of app
* Proceeding to the "Popular" category on the home page.
* I opened the app from the homescreen and went to popular local businesses.

Application Transition:



Selected Task Label:

find directions for Newell Simon Hall

Resources Used:

phone number, exact location, and approximate location

Crowd expected resource use:

Yes

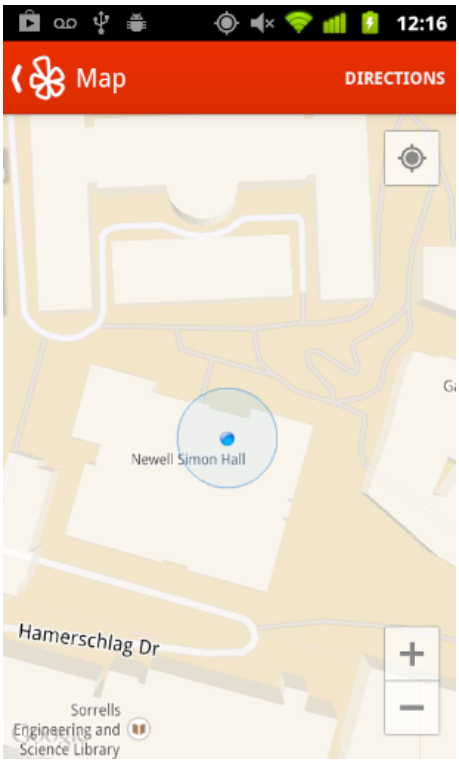
Crowd comfort [-2,2] (N, (μ,σ)):

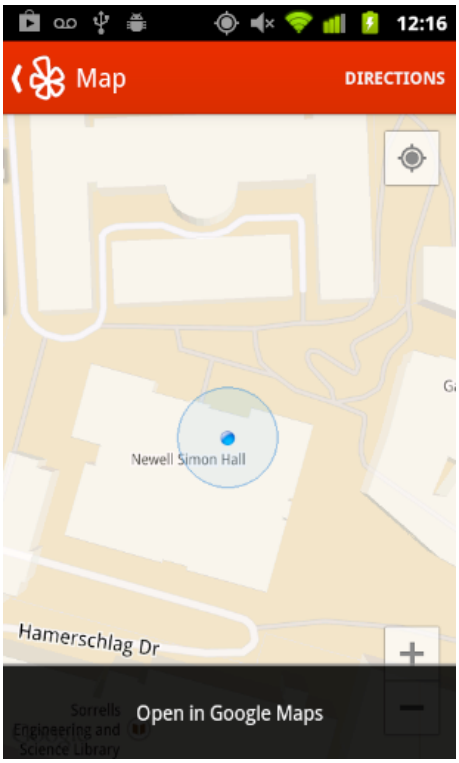
N=?, (μ=-0.4, σ=1.1)

Crowd-sourced Task Labels:

* directions to a place
* finding restaurant locations and directions on a map
* Find more information on the point plotted.
* find food in my area
* look for right place
* find directions for Newell Simon Hall
* Getting directions to a location
* directions to Newell Simon Hall
* Selecting a business to view in Google Maps

Application Transition:





Selected Task Label:

find restaurant best match

Resources Used:

phone number, exact location, and approximate location

Crowd expected resource use:

Yes

Crowd comfort [-2,2] (N, (μ,σ)):

N=9, (μ=1.3, σ=0.5)

Crowd-sourced Task Labels:

* Find a Best Match Restaurant nearby
* Finding places to eat.
* find a good restaurant nearby
* finding closest restaurants
* find restaurants near me
* Narrow your searches
* Trying to find a restaurant and sort the search results.
* find restaurant best match
* search for restaurants

Application Transition: