

Towards Scalable Evaluation of Mobile Applications through Crowdsourcing and Automation

Shahriyar Amini, Jialiu Lin, Jason Hong, Janne Lindqvist, Joy Zhang

February 29, 2012

[CMU-CyLab-12-006](#)

[CyLab](#)
Carnegie Mellon University
Pittsburgh, PA 15213

Towards Scalable Evaluation of Mobile Applications through Crowdsourcing and Automation

Shahriyar Amini^a, Jialiu Lin^a, Jason I. Hong^a, Janne Lindqvist^b, Joy Zhang^a
Carnegie Mellon University^a Rutgers University^b
Pittsburgh, PA New Brunswick, NJ
shahriyar@cmu.edu janne@winlab.rutgers.edu

ABSTRACT

With the widespread adoption of smartphones, mobile applications have gained mainstream popularity. However, the potential privacy and security risks associated with using mobile apps are quite high, as smartphones become increasingly integrated with our lives, being able to access our email, social networking accounts, financial information, personal photos, and even our cars and homes. To address this problem, we introduce AppScanner, an automated cloud-based service based on crowdsourcing and traditional security approaches to analyze mobile applications. Considering the large and growing number of mobile applications, our envisioned service builds on crowdsourcing, virtualization, and automation to enable large-scale analysis of apps. AppScanner provides end-users with more understandable information regarding what mobile apps are really doing on their devices. This paper offers an overview of our vision for building AppScanner, as well as work to date in specific components, including automated traversal and monitoring of mobile applications, and interactive visual presentation of app traversal results. Armed with transparent and descriptive information regarding app behavior, users can make better decisions when installing and running apps.

Author Keywords

Crowdsourcing, Mobile Privacy and Security.

ACM Classification Keywords

H.5.2 Information interfaces and presentation (e.g., HCI): Miscellaneous.

General Terms

Experimentation, Human Factors, Security.

INTRODUCTION

Mobile app stores have responded to consumer demand for mobile applications, with the number of application downloads well into the billions. As of March of 2012, the Apple App Store offers more than 550,000 applications, with application downloads topping 25 billion [3]. Similarly the Google owned Android market offers Android users over 400,000 applications [24]. Considering the omnipresent availability of context-aware computing resources and users' eagerness to install and use apps, mobile devices have become an attractive platform to offer desirable services. However, the same appealing features that enable the offering of services also simplify the collection and inference of private

information regarding users. For instance, prior to user criticisms, Path, a smart journal application with sharing features, uploaded users' entire contact lists to its servers [34]. Third party access to sensitive user information leads to potential opportunities to exploit users, which have been exercised in the recent past for physical stalking [10] and home invasions [36].

The application markets 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 [29]. On the other hand, the Android Market relies on the developers to specify the resources used in an application manifest and directly reports this information to the end user [18]. However, it is questionable whether users have the knowhow to clearly comprehend how such resources can be used in malicious ways [15].

Prior work by security experts and the research community have raised concerns about misbehaving mobile apps [1, 2, 11, 13, 22, 30, 31, 34]. These works have been successful in exposing the privacy and security related risks involved with using mobile applications and have also raised awareness through media coverage. However, expert knowledge is required to set up and perform the techniques used in these prior efforts and also to digest the results obtained. Furthermore, although some of the approaches are effective in detecting the use of sensitive information [11, 13], it is not clear when the use of the sensitive information by an app is both legitimate and beneficial to the end-user.

Consider two applications, a blackjack game and a restaurant finder, both of which request user location, a sensitive piece of information regarding the user. Although prior techniques can detect the use of location, they would fail in marking the use of location as legitimate or not. As such, human intervention, and in some cases, expert knowledge, is necessary to mark the use as legitimate or not. A Wall Street Journal study of mobile application behavior [35] and also an independent study of mobile application data usage [23] 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. Considering the large number of mobile applications available on app markets, manual inspection of app behavior is not feasible.

To determine the legitimate use of sensitive information by applications and also to address the scalability problem, we envision a new cloud-based service, AppScanner, which relies on crowdsourcing, virtualization, and automation to evaluate mobile application behavior. Specifically, AppScanner relies on the use of automation and traditional security techniques to learn application behavior in fine granularity. AppScanner 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 and the use of automated techniques, we propose that AppScanner is a scalable approach towards analyzing mobile applications and providing large coverage of app markets. Further, we conjecture that by relying on the wisdom of crowds, AppScanner can produce application behavior evaluations that would be close to expert analysis of the app behavior.

AppScanner is our vision of how large-scale evaluation of mobile app behavior could take place. As such, not all the ideas in this report have been implemented, and therefore, some aspects of the system remain to be evaluated. However, we consider the proposed approach presented here as one viable route to achieving large-scale evaluation of mobile applications.

APPSCANNER

AppScanner is comprised of multiple subsystems which work together to evaluate applications and to produce understandable summaries of application behavior for end-users. For the purposes of this work, we will focus on the Android operating system. We made this decision based on Android being open source as well as the availability of tools and extensions for Android (in particular, TaintDroid [13], and TEMA [32]).

Figure 1 shows a high-level architecture of our proposed approach. To this end, we are developing four major subsystems. The *App Mapper*, which we have named *Squiddy*, analyzes a given mobile app and outputs a control flow graph of major screens, associated privacy-related behaviors of the app, and major tasks that can be done with the app. At present time, we have a working prototype of Squiddy. We are also working on an interactive visualization, Gort, which presents the application behavior information obtained by Squiddy, to enable easier analysis and better understanding of apps.¹

¹Squiddy refers to the scanning portion of AppScanner, named after robot Sentinels from *The Matrix*. Gort refers to the AppScanner GUI, named after the robot in *The Day the Earth Stood Still*.

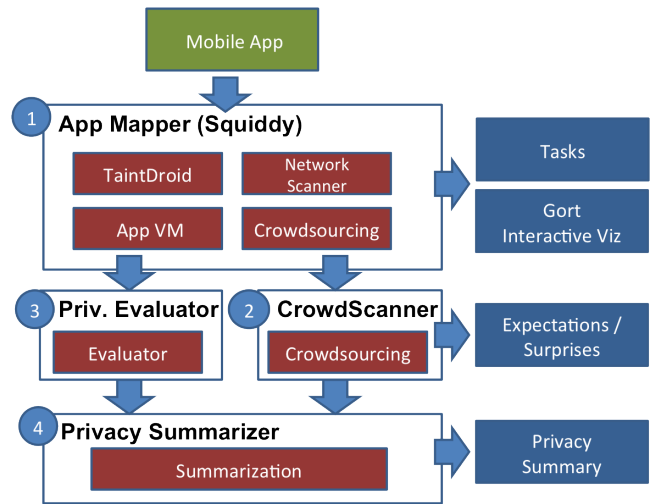


Figure 1. Android App Scanner system diagram, showing major subsystems and their outputs. The App Mapper subsystem, Squiddy, takes an app and traverses the main screens of the app, monitors what resources are being used and what remote servers are being contacted. We envision that App Mapper can also crowdsource what tasks can be done on various screens. As part of the App Mapper, we have developed Gort, which is an interactive visualization for the results obtained by Squiddy for a specific application. The CrowdScanner takes the tasks and the results obtained by Squiddy, and captures people’s expectations and surprises regarding the application behavior. The Privacy Evaluator evaluates what can be inferred by third parties based on the app’s type, frequency, and access pattern of the mobile device’s resources and the user’s information. The Privacy Summarizer combines the Privacy Evaluator’s inferences about potential privacy leaks and people’s perceptions and concerns produced by the CrowdScanner to distill a compact and understandable privacy summary for the application.

The other subsystems of AppScanner have yet to be developed. However, the *CrowdScanner* takes the results from Squiddy, and uses crowdsourcing techniques to capture what kinds of privacy concerns and surprises people have, both at a coarse granularity (e.g. flagging that the LED flashlight app requests Internet access) and a fine granularity (e.g. flagging unusual behaviors on specific screens and tasks). The *Privacy Evaluator* estimates what kinds of things a third party can infer based on the information they can get through an app. The *Privacy Summarizer* collects people’s perceptions and generates a simple-to-understand summary of a mobile app’s privacy-related behaviors. Figure 2 shows a mockup of potential privacy summaries generated by the Privacy Summarizer.

App Mapper

The goal of the App Mapper is to design, implement, and evaluate a tool to map out the major screens, associated privacy-related behaviors, and most common tasks for a given app. To this end, we have developed a preliminary prototype of mobile app traversal software, Squiddy, which maps major screens, takes screenshots, and monitors resources and sensitive information used by the application. We are also working on an interactive visualization, Gort, that enables graphical analysis of the results obtained through Squiddy. The primary outputs of the App Mapper subsystem are the inter-

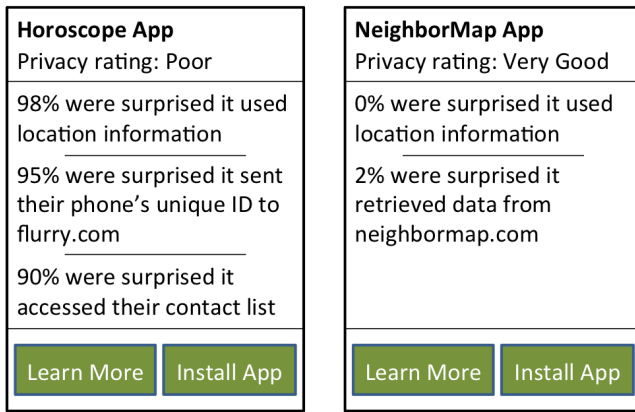


Figure 2. Example mockups of privacy summaries that focus on showing unexpected behaviors of apps. These privacy summaries would be generated through automated app scanning techniques combined with crowdsourcing.

active information visualization provided by Gort, a control flow graph of the major screens of an app (along with resources used and data transmissions), and a list of common tasks along with associated screens (remaining).

Learning about Apps

The crowd's opinion with regards to application behavior can be obtained in two ways. The first approach is to present the crowd with a coarse-grained view of the application and also the high level resources that it uses. This can be done relatively easily for Android applications as developers are required to provide a description of the application as well as the resources that the application uses prior to placing it on the Android Market [18]. Although obtaining the crowd's opinion may be relatively easy, it may not always be exactly clear why an application uses a certain resource. For instance, a black jack app could use the phone's unique ID and Internet access to present a ranking of scores to gamers. However, this might not be immediately obvious to the crowd through just the application description and list of resources.

Another approach is to provide the crowd with more fine-grained information about what the application is actually doing. This way the crowd can make more informed decisions regarding why the app is using certain resources. To achieve this, it is necessary to both explore the app and also to monitor what resources the app uses and which servers the app communicates with while running. AppScanner would then provide the crowd with screenshots of application screens, along with what resources are simultaneously used by the app on each screen. Also, through app traversal, AppScanner generates a graph of the application which illustrates how application screens are linked together, resulting in a coarse-grained Control Flow Graph (CFG). In order to explore the application, one method would be to traverse the application from its main screen using *breadth-first search (bfs)*. We have prototyped a proof-of-concept version of the traversal software, Squiddy, using bfs and TEMA [32], a model-based testing tool for Android GUIs. Using TEMA, we can get the clickable widgets on the current screen (us-

ing Android's window APIs), and simulate clicking on those widgets (using Android's monkeyrunner APIs). For simple Android apps, we can already generate a graph of the screens that users see on a real phone (see Figure 3). For an actual deployment, we plan on running this traversal in a virtual machine to make it easier to scale up as well as monitor resources. Each Squiddy run would be isolated in its own virtual machine environment.

In addition to traversing the application and taking screenshots, it is important to monitor what resources are being used by the application on each screen. For this purpose, one could monitor application resource usage through a network analyzer as is done by the Wall Street Journal study of mobile applications [35] and Hunt's independent expert study of mobile application data usage [23]. However, the issue with just using network transmission analyzers is that unless the data is unencrypted one cannot make a guess as to what sensitive information is being sent out. Also, parsing the mobile apps network requests are non-trivial. As such, we rely on dynamic information tracking, specifically Taint-Droid [13], to monitor application resource usage. Taint-Droid logs when sensitive information is transmitted as well as identifies the type of sensitive data involved.

Identifying Application Tasks

Although it is useful to know which screens of the application are responsible for a particular network transmission and the potential transmission of sensitive information, it would be even more valuable to map the transmission of sensitive information to specific tasks that can be performed by an app. Moreover, some network transmissions may be the result of multiple interactions on a screen or several screens of the application. Therefore, it would be more exact to associate the transmission with a task, rather than a single screen. To tackle this problem, we propose the use of information extraction (IE) techniques as well as the crowdsourcing to identify tasks that can be performed using the app. To elicit these tasks, one approach would be to use IE to identify verbs in the text description of the app, which as noted previously are available on the Android Market. A second approach is to give this same description to crowdsourced workers, and ask them to list different kinds of tasks they think can be done. Finally, a third approach would be to provide the coarse-grained CFG of the application, as generated by Squiddy, to the crowd along with screenshots, and ask what task a particular path through the graph represents. Using a combination of these techniques, tasks can be mapped to paths along the CFG, which can then be mapped to specific network requests and application behaviors.

Visualizing Apps

It can be difficult for app developers and privacy analysts to understand the behavior of apps. To speed up the process of analysis, both for others and for ourselves, we are developing Gort, an interactive tool that shows 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. At present time, Gort is still a prototype. However, using Gort, an analyst can see the over-

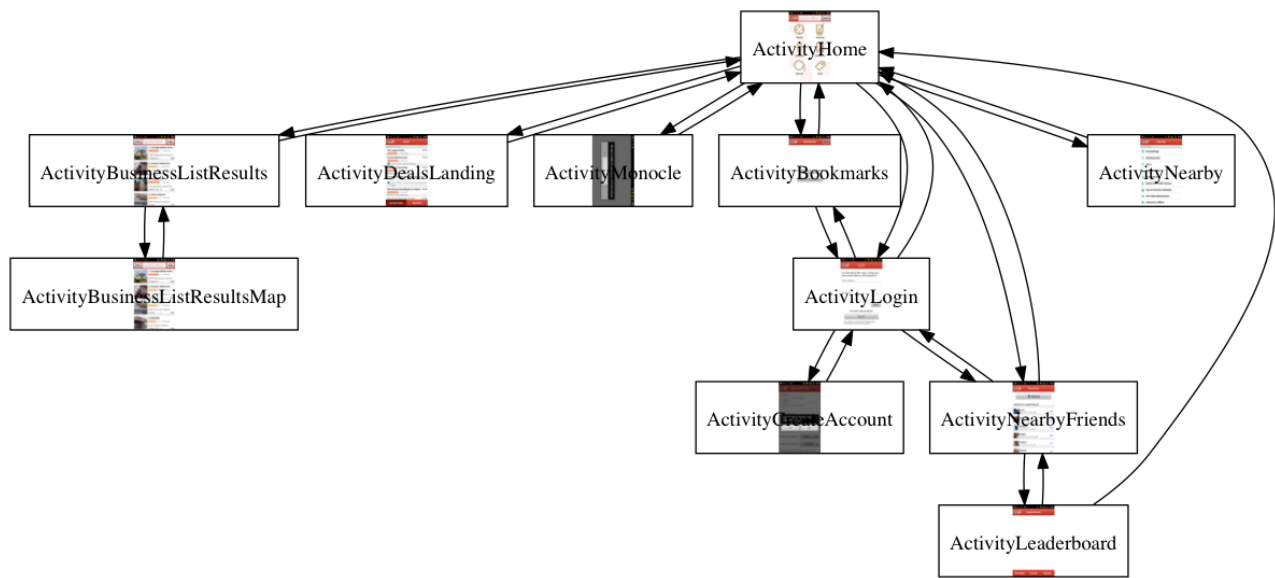


Figure 3. Example of a control flow graph of screens that our early prototype can generate. This figure shows major screens in the popular Yelp application. Our prototype also captures actual screenshots of screens that users see. A thumbnail version of the corresponding application screen is shown inside of each node.

all behavior of their 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 [4]. For example, our user might select a resource (e.g. Contacts List) and see what screens use that resource. She might select a remote server and see all the screens where data is sent to the server plus what data is sent. Rather than just seeing raw bytes, she can see highlighted strings that indicate when the phone’s unique ID or contact list is shared. Lastly, she might select a task and see associated screens, resources, and remote servers.

The majority of the techniques described in the App Mapper section have been developed and briefly evaluated. The following descriptions of the remaining subsystems presents the missing pieces that we envision would enable the automated evaluation of mobile application behavior.

CrowdScanner

The CrowdScanner subsystem is responsible for capturing people’s perceptions of how an application is behaving. As previously mentioned, people’s understanding of the application could be based on a high-level view of the application, perhaps just the application description and permissions or resources uses. However, to get fine-grained information about what an application is really doing, it is necessary to provide the crowd with more detailed information about how the application works.

The main idea here is to provide the crowd with fine-grained information that is previously generated by Squiddy. There are many ways of presenting the information obtained through Squiddy. For instance, the crowd could be provided with a single screenshot of the application along with requests and sensitive data transmissions on the corresponding screen of

the app. Another approach would be to isolate a task that can be performed using the app and present the crowd with the chain of screens involved, the requests and the sensitive information transmitted pertaining to the particular task. We want to capture the crowd’s understanding of the inner workings of the application, their surprises, and their perceptions of how the application should behave given its features. For instance, we are interested to know if the crowd can identify that to obtain nearby points of interests from a search application such as Google Maps, the transmission of the user’s location may be an acceptable application behavior.

The crowd’s perception of an application may not be very accurate given the technical nature of the topic. To evaluate the crowd’s responses regarding application behavior, we would compare the crowd’s answers to answers from experts for a number of applications. Our goal here is to see how close the crowd can get to expert answers, how much compensation is necessary, and also how to automate and optimize the crowdsourcing process to get accurate responses in a short amount of time with a reasonable amount of compensation.

Privacy Evaluator

Depending on what type of sensitive information an app accesses, how often, and the access pattern, third parties may be able to infer different amounts of private information. For instance, an application that samples the user’s location once every second cannot only tell about the user’s current location, but can also infer the user’s driving habits, home location, and other significant places. On the other hand, if an application only requests the user’s location once it starts, then the level of inference that the application provider or a third party can make is drastically reduced.

The problem here is that even though non-experts may be able to identify the use of sensitive information by an application as normal or abnormal, they may not have the required knowledge to know what can really be inferred, specially when technical inference can accomplish much more than expected. Recent work by Brush et al. shows that even though non-experts can identify location as a sensitive piece of information, they are not always able to understand what can be inferred using technical methods [7].

To address the aforementioned problem, we propose a privacy evaluator subsystem, which quantifies the personal information an app can infer, using the results from Squiddy's traversal. The Privacy Evaluator will also present its results through scenarios that may be more understandable to an average user, such as "this app can figure out the location of your home and office", or "this app can find out that you are checking emails when you are driving". Prior work by Enck et al. shows that rules based on Android permissions can be used to identify possibly dangerous combination of Android permissions [14]. Similar rule based analysis which considers the type, frequency, and pattern of sensitive information accesses can help inform users regarding the potentially intrusive inferences that third parties can make using app data.

Privacy Summarizer

The final AppScanner subsystem is the Privacy Summarizer which offers end-users privacy summaries that are automatically generated from the results of the CrowdScanner and Privacy Evaluator. The final output of our work, and the one that will be most visible to end-users, is the privacy summaries that will help people more accurately understand the privacy-related behaviors of a mobile app. Figure 2 shows mockups of such privacy summaries. The summaries will include what the crowd deemed to be normal and abnormal for a particular application given the tasks that the application can perform. This information will also take into account the inference that can be made based on data collection from the application, as decided by the Privacy Evaluator. We will design and evaluate our privacy summaries over multiple iterations. We will start with semi-structured interviews with experts and novices to understand people's understanding of privacy as well as concerns over mobile apps. We will also rapidly prototype several mockups of privacy summaries, helping us to home in on the most effective way of presenting these summaries.

IMPLEMENTATION

At present, the full implementation of AppScanner is in progress. Currently, we have prototyped the App Mapper, Squiddy, and the interactive visualization tool, Gort. Here we focus on the implementation aspects of Squiddy and Gort.

Squiddy

Squiddy is responsible for traversing mobile applications and monitoring the resources that are used. We achieve this task by building a *breadth-first search* traversal on top of TEMA [32] and monitoring the use of sensitive information by instrumenting applications using TaintDroid [13]. Squiddy

currently traverses Android applications by clicking on potentially interactive elements of the app UI. As the App Mapper discovers more screens of the app, it continues digging deeper into the application. The TaintDroid logs of sensitive information usage have timestamps which are then associated with interactions on a specific screen after the traversal is completed. There are a number of limitations to this approach which are discussed in §Research Challenges.

Post-traversal Analysis

Squiddy builds a database of all the UI interactions, sensitive information usage, and external server requests of a particular application. Following this step, there is a post-traversal processing stage that combines all the available information on transmissions of sensitive information to particular screens of the app. The processing stage also compiles a list of all servers contacted and all sensitive resources used by the app. During the post-processing, a CFG diagram of the application, which can be viewed and interacted with in Gort, is generated programmatically using Graphviz [19].

Gort Visualization

Gort presents the outputs from the Squiddy traversal and post-traversal processing in the form of an interactive GUI. 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 for developers and privacy experts.

Figure 4 shows a screenshot of the Gort UI presenting information regarding the Yelp mobile app. The Yelp app allows users to search for nearby points of interests, such as restaurants, check in to places, and find deals². On the top left hand corner of Gort, the user is presented with a control flow graph of the app. The control flow graph is interactive. Upon receiving a click on a single activity or screen, the app presents a screenshot of the activity on the top right portion of the app. In the midsection, the GUI shows corresponding requests and request details of the selected screen. The application description and permissions, as obtained from the Android Market, are shown on the bottom right portion. Finally, the lower left and lower mid portions of the UI show all the servers and sensitive informations 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 requests related to servers x or y).

²www.yelp.com

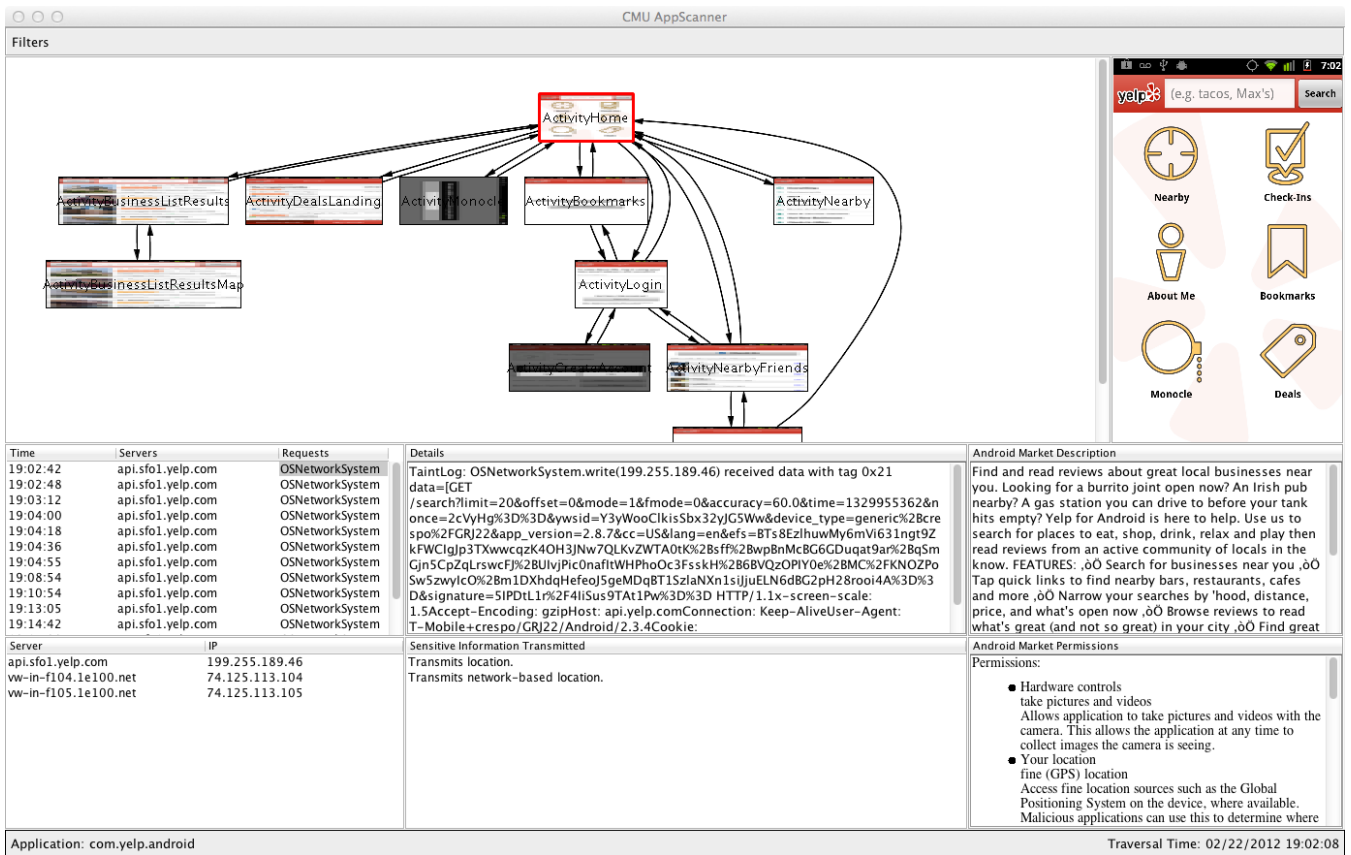


Figure 4. The AppScanner GUI, Gort, enables application developers and privacy experts to learn about an application’s behavior through an interactive GUI. The Gort UI, as depicted here, presents an overview of the Yelp application, which includes a high-level control flow graph of the Yelp screens, 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 control flow graph. The GUI also presents the description of the application as available on the Android Market along with the permissions declared by the developer. The user 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.

The GUI offers another means of information presentation through brushing and linking [4]. Using brushing and linking, the user can select a particular screen and see all related pieces of information highlighted. For instance, clicking on a particular activity highlights all the servers the particular activity uses and also highlights all the resources that particular activity requests. The GUI has two modes of use, filter, and brush and link. Depending on the type of visualization techniques that best tailors to the user’s task, the user can select her desired mode.

EVALUATION

We plan to evaluate each subsystem of AppScanner individually. Each subsystem can be optimized and evaluated separately. Nevertheless, the success of the entire system also depends on how well the subsystems integrate. The evaluation for the Privacy Summarizer would also be a high level evaluation for the AppScanner as the final output of the system is the privacy summarizes for mobile applications. Here we describe the evaluation or proposed evaluation for each subsystem.

App Mapper

The App Mapper produces information regarding the application traversals, monitors application resource usage, and also enables dynamic interaction with the results through the App Map visualization. Presently, we evaluate the App Mapper by manually inspecting the results from Squiddy and also studying how privacy experts interact with the App Mapper output through Gort. We ask the experts regarding their experience with Gort, its ease of use, and the quality of information generated. We also ask them to flag information that Squiddy may have incorrectly associated with the app or unexpectedly assigned to a screen in the app. We then manually check the results from the flags and the app to see how close Squiddy is to the ground truth. In the future we plan to release Squiddy and Gort as privacy analysis tools. This would enable us to get feedback from a large audience.

CrowdScanner

CrowdScanner results can be verified by comparing the perception of the crowd with those of privacy experts. The evaluation should determine whether the crowd’s response can approach the same levels as the experts. As crowdsourcing

people’s perceptions regarding mobile applications is an important idea of AppScanner and also one of the fundamental components in generating Privacy Summaries, it is important to verify that the crowd can actually match experts’ opinions regarding application behavior.

Privacy Evaluator

One evaluation method for the Privacy Evaluator would be to compare its predications regarding potential third party inference to a baseline. The baseline here would be what experts would predict as potential third party inferences based on the input available from the App Mapper. Here the same input would be made available to the Privacy Evaluator and the experts. There are two methods of comparison. One method is to look at the precision and recall of inference that experts would expect third parties to be able to make with Privacy Evaluator suggestions. The other method is to present the Privacy Evaluator results to experts and ask them if the suggestions are plausible.

Privacy Summarizer

The primary test for success is if our privacy summaries can help people make better trust decisions regarding the installation and use of a given mobile app. However, it is difficult to say for sure if someone made the correct trust decision, since people have different values and thresholds for risk. Instead, we will operationalize this notion of better trust decisions using the C-HIP framework [37]. The C-HIP model is taken from the field of human factors and offers a framework for analyzing the effectiveness of warnings. Pragmatically, this framework asks if people notice the warning, comprehend it, believe it, and are motivated to act. We have successfully used this framework in the past to analyze the effectiveness of anti-phishing warnings [12], the results of which were eventually adopted in Microsoft’s Internet Explorer 8.

RESEARCH CHALLENGES

Considering that AppScanner is comprised of four subsystems, each involving various complexities, a number of challenges exist. Here we present the challenges for each one of the subsystems and also present potential solutions.

App Mapper

There are many technical challenges in traversing mobile apps. Examples include handling username and passwords, text input fields, scrollable areas, location-based content, and screens that require some form of input and acknowledgment (e.g. Ok or Next). There are also several challenges that we currently do not plan on addressing, in particular those that have the potential for exponential blow-up. These include managing configuration options or handling user-generated lists (such as a playlist).

Complex Interactions

We will develop new techniques for traversing more complex apps. For instance, we will leverage design patterns to recognize common genres of screens, such as login screens

(which have words like username, password, sign-in, or login) or forms screens (multiple input fields and buttons labeled previous, continue, or next). Using these design patterns, we can enable more thorough traversals as well as recognize what kind of input may be necessary to continue onto other screens.

We will also explore the use of crowdsourcing to provide input to apps. For instance, for an app that shows nearby points of interest (such as Google Maps and Yelp), we might ask the crowd for a sample input to a text field. The crowd could then offer the word ‘restaurant’ for a text field requiring a search query. Lastly, we will experiment with simple computer vision techniques. For example, we currently use basic computer vision to recognize when a software keyboard is shown, letting us know we should hit the back button and remove it from our screenshot. As another example, some apps use the app’s icon to get back to the home screen. Other examples where computer vision may be useful include identifying non-standard buttons and recognizing common kinds of layouts.

Other difficulties include complex application interactions that involve multiple user actions across one screen or several screens of the application. Once Squiddy has evolved to the point where it can easily cover single interactions, we plan to further take advantage of crowdsourcing to identify tasks that our control flow graphs do not cover. The successful traversal of complex interactions would most likely require multiple responses from the crowd spanning finding of complex interactions, providing input to them, and verifying that the input successfully resulted in generating a response from the app. Breaking a complex task down to multiple smaller tasks and feeding output from one task to the next have been explored by prior work. Bernstein et al. present one such work in the domain of word processing [6]. We propose using similar techniques to further provide coverage of application traversal.

Resource Monitoring

Resource monitoring is presently performed through the analysis of TaintDroid logs and assigning sensitive network transmissions based on timestamp. Based on our experience, this seems to work well. However, there are times where a sensitive transmission is incorrectly assigned to the application being analyzed. For instance, as we traversed the Yelp application we noticed sensitive transmissions to a Google owned server. This was hard to replicate and most likely the transmission is associated to background services running on Android rather than triggered through Yelp. Our short term solution to this problem is to enable Gort users to flag potentially abnormal network communication as incorrect detected. In the long term, we believe that such problems can be detected through merging the Squiddy outputs of several traversals. In addition to incorrectly assigned network transmissions, we would like to identify network transmissions as user-triggered, periodic, and random. Similar to the problem of abnormal network transmissions, we believe that combining the results from multiple App Mapper results would allow us to further categorize network transmissions.

Such categorization allows for better input to the CrowdScanner and Privacy Evaluator subsystems which would result in better privacy summaries for applications.

Coverage

App Mapper coverage can be described with respect to an Android Application and also with respect to the Android Market. Within an application, coverage would describe how much of the application's activities, sensitive information requests, and network transmissions get discovered and exercised by Squiddy. This depends on how successful Squiddy is in providing input to the application and monitoring its resources. Without having the actual source code of the application under analysis, within app coverage is difficult to calculate. However, given information regarding the number of permissions the app requests and the number of activities and services that the application provides, an approximate coverage percentage could be calculated based on how many of the permission uses are detected and how many of the activities and services have been discovered.

Coverage with respect to the Android Market may be simpler to calculate based on the number of applications on the market. As previously mentioned, AppScanner currently focuses on scanning apps with simple GUI forms where the UI elements providing interaction can be easily detected and perturbed. Therefore, complex apps, such as games, are outside of the current scope of AppScanner. Further, as mentioned, it is not clear how much coverage Squiddy can provide within an application. To consider an app coverage to be successful, a coverage threshold should be defined. Assuming the Squiddy coverage passes the coverage threshold, the app would be counted towards Squiddy Android Market coverage. How to define the coverage threshold is another problem which has to be addressed.

CrowdScanner

A number of challenges exist when using crowdsourcing to complete research tasks. It has yet to be decided whether the crowd can recognize privacy implications of a mobile app in the same way that experts in the field can. However, to address this question, we are conducting a number of surveys from experts and crowd members. In the long run, we would like to optimize crowd response with respect to accuracy obtained, and time and compensation required. As the CrowdScanner relies on input from the App Mapper, it is critical that only correctly detected and associated sensitive requests are passed on to the CrowdScanner. Therefore, any sensitive requests which are potentially incorrectly assigned to the app should not be passed on as input.

Another challenge with respect to crowdsourcing people's perceptions of mobile application behavior is the irregularities associated with what is normal vs. abnormal application behavior. For instance, it is common for app developers to monetize free apps through presenting ads to the users. In doing so, some apps share the user's private information with third party ad providers to present more relevant ads. The private information could be any combination of the user's age, gender, location, unique identifiers (e.g., phone

number, device serial number, IMEI), etc. Although to an expert presenting ads and sharing private information with the ad provider might seem normal for free apps, this may not necessarily be the case for non-technical crowd workers.

Privacy Evaluator

The Privacy Evaluator subsystem is relatively straightforward. As long as correct rules and inference techniques are used to recognize potential private information inferences that third parties can make, the privacy evaluator can provide correct results. Similar to the CrowdScanner, since the privacy evaluator relies on data from the App Mapper, it is critical that only correctly detected and associated sensitive requests are passed on to the Privacy Evaluator.

Privacy Summarizer

There are several design issues related to providing privacy summaries. One example issue is examining when it is better to show unusual coarse-grained behaviors (e.g. this LED flashlight app requests Internet access) versus fine-grained ones (e.g. this app sends your contact list data to xyz.com when on these screens). Another example issue is how well an overall privacy score may or may not help with people's understanding. A third issue is prioritizing which behaviors are shown first. For example, is it better to prioritize by biggest surprises (e.g. 90% of crowd participants were surprised by app behavior x) or by resource type (e.g. always prioritizing contacts list and location data over all other types)?

Another major design issue here is when and how to best show the privacy info to people. For example, we can show variants of our privacy summaries to people when searching for an app, when examining an app, right before installing an app (the current design on Android), as well as after an app is installed. Past work in evaluating EULAs [17] found that showing information right before install time didn't work at all. Other past work found that showing elements of privacy results when searching for apps did have an effect on people's behaviors on e-commerce sites [16].

RELATED WORK

AppScanner builds on a number of different topics. As such, we cover prior work from a number of different areas in this section, specifically, work related to privacy and security, crowdsourcing, and user interfaces.

Privacy and Security

A number of prior works explore privacy leaks. Privacy Oracle finds potential data leaks using black box differential testing and running an application under different settings [25]. The WiFi Privacy Ticker displays information about sensitive data sent from the computer to the network [9]. TaintDroid [13] and PiOS [11] are two works that take advantage of dynamic and static analysis of mobile apps, respectively, to find potential leaks of sensitive information. AppScanner is similar in nature in that we attempt to detect sensitive information leaks in mobile applications, however, our main goal is to be able to present this information

in an understandable fashion to end-users. Static and dynamic analysis of apps have limitations and can be bypassed if the application developer truly wants to be malicious [8, 28]. As we use TaintDroid to analyze application resource usage, AppScanner also faces the same challenges. However, AppScanner’s main focus is to inform end-users about application behavior of mobile apps in general, especially in cases where the apps are not intentionally malicious. In this sense, AppScanner is more similar to the work on privacy nutrition labels by Kelley et al., which offers simple visualizations to help people understand why a website collects private information [26]. Nonetheless, AppScanner is different from privacy nutrition labels in that we use crowdsourcing to obtain information regarding application behavior and look specifically at mobile apps. Further, we aim to automate the end-to-end process using traditional security approach and crowdsourcing. A number of works present methods to blocking data leaks to the network or faking the capabilities of the phone or the information provided to third parties [5, 22, 38]. However, our goal is not to manipulate information flow to third parties, but rather to present end-users with understandable information about what the application is really doing.

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 we will use in our research. Most tasks on MTurk do not require special skills and tend to involve small payments. MTurk has been successfully used in a number of projects, including for example automating online tasks (such as image labeling) and user studies [27]. Crowdsourcing has also been used in the past to evaluate visualization design [20]. Perhaps the most sophisticated work in crowdsourcing is Soylent [6]. Soylent uses multiple ways of structuring tasks and organizing people so as to yield reasonably good results for relatively high-level tasks. This work showed how certain kinds of design patterns in organizing workers could yield good results.

Our research uses crowdsourcing as one of its components but also builds on past work in several ways. First, our work will develop new ways in applying crowdsourcing to computer privacy. To the best of our knowledge, our work would be the first in using crowdsourcing directly for privacy. Second, our work will look at new ways of combining automated techniques with crowdsourcing, including for example capturing perceptions of privacy for a given app (coarse granularity) as well as specific screens and sequences of screens (fine granularity). Third, our work will also investigate new ways of doing effective divisions of labor, as well as ways of aggregating results to yield semantically useful results.

User Interfaces

There has been increasing research in usable privacy and security, looking at how to make tools and UIs useful and understandable for lay people. However, thus far, there has been relatively little work in this field examining mobile

apps or interfaces for specifying what personal information a person is willing to share. Perhaps the most relevant work on the usability of permissions screens is an unpublished tech report at Microsoft Research looking at Facebook permissions [33]. This work focused on how visual representations of resources could help people make better decisions. Our new idea here is to focus instead on unexpected uses of resources by apps (rather than showing all resources), and to raise the level of abstraction by also framing things in the context of tasks rather than just resources alone. WebQuilt is a web logging and visualization system that helps web design teams run usability tests (both local and remote) and analyze the collected data [21]. AppScanner visualization will build on some of the WebQuilt concepts to visualize application resource usage.

Our work on improving privacy summaries will build on the privacy nutrition labels work [26]. Our designs will make use of these findings as well as methods used for evaluation; however, the nutrition labels assume an optimistic case where web sites voluntarily disclose accurate information. In contrast, we assume a pessimistic case, using a number of techniques to extract behavior as well as crowdsourcing for gauging people’s perceptions of those behaviors.

SUMMARY

In this report, we presented our vision on how to enable the large scale evaluation of mobile application behaviors through crowdsourcing and automation. The number of applications on the Apple App Store and Android Market have collectively reached over a billion. Furthermore, the number of application downloads are already well into the tens of billions. As mobile apps have access to both mobile device sensors and also users’ personal data, it is critical for users to know how mobile apps are using sensitive information and resources on their devices. However, considering the large number of mobile apps, mobile application behavior cannot be manually inspected for each application. Furthermore, it is necessary to transform the outputs of traditional privacy and security approaches to language that is both understandable and informative to users. To this end, we presented a cloud-based service, AppScanner, that uses crowdsourcing and automation to address the scalability problems involved in analyzing lots of applications and that also divides up the evaluation process such that its output can be presented to end-users in an understandable fashion.

At present time, we have developed two components of AppScanner, Squiddy and Gort. Squiddy enables automated traversal of mobile apps along with resource usage and sensitive information transmission monitoring. Gort is a by-product of our development process, which is used to speed up the development of AppScanner through visualizing Squiddy output. Gort can also be used by privacy experts and developers to understand mobile app functionality. We are currently working on implementing and evaluating the rest of our proposed architecture. Our end-to-end cloud-based service should enable users to make informed decisions about installing and running applications. We conjecture that the successful implementation and adoption of AppScanner would

enable users to obtain applications which not only provide desired features but also meet users' privacy expectations.

REFERENCES

1. C. Albanesius. Congress Asks Apple for Details About Address Book Privacy. *PCMag*, Feb 2012. <http://www.pcmag.com/article/print/294178>.
2. A. Allan and P. 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>.
3. Apple Inc. Apples App Store Downloads Top 25 Billion. *Apple Press Info*, Mar 2012. <http://www.apple.com/pr/library/2012/03/05Apples-App-Store-Downloads-Top-25-Billion.html>.
4. R. A. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.
5. A. R. Beresford, A. Rice, N. Skehin, and R. Sohan. MockDroid: trading privacy for application functionality on smartphones. In *Proceedings of HotMobile 2011*. ACM, 2011.
6. M. S. Bernstein, G. Little, R. C. Miller, B. Hartmann, M. S. Ackerman, D. R. Karger, D. Crowell, and K. Panovich. Soylent: a word processor with a crowd inside. In *Proceedings of the 23rd annual ACM symposium on User interface software and technology*, UIST '10, pages 313–322, New York, NY, USA, 2010. ACM.
7. A. B. Brush, J. Krumm, and J. Scott. Exploring end user preferences for location obfuscation, location-based services, and the value of location. In *Proc. of UBIComp*, 2010.
8. L. Cavallaro, P. 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.
9. S. Consolvo, J. Jung, B. Greenstein, P. Powledge, G. Maganis, and D. 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.
10. Dailynews. How Cell Phone Helped Cops Nail Key Murder Suspect Secret Pings That Gave Bouncer Away. *Dailynews*, Mar 2006.
11. M. Egele, C. Kruegel, E. Kirda, and G. Vigna. PiOS : Detecting privacy leaks in iOS applications. In *NDS '11*, 2011.
12. S. Egelman, L. F. Cranor, and J. Hong. You've been warned: an empirical study of the effectiveness of web browser phishing warnings. In *Proceedings of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, CHI '08, pages 1065–1074, New York, NY, USA, 2008. ACM.
13. W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI '10, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association.
14. W. Enck, M. Ongtang, and P. 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.
15. A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner. Android Permissions: User Attention, Comprehension, and Behavior. Technical Report UCB/EECS-2012-26, University of California at Berkeley, 2012.
16. J. Gideon, L. Cranor, S. Egelman, and A. Acquisti. Power strips, prophylactics, and privacy, oh my! In *Proceedings of the second symposium on Usable privacy and security*, SOUPS '06, pages 133–144, New York, NY, USA, 2006. ACM.
17. N. Good, R. Dhamija, J. Grossklags, D. Thaw, S. Aronowitz, D. Mulligan, and J. Konstan. Stopping spyware at the gate: a user study of privacy, notice and spyware. In *Proceedings of the 2005 symposium on Usable privacy and security*, SOUPS '05, pages 43–52, New York, NY, USA, 2005. ACM.
18. Google Inc. Security and Permissions. *Android Developers*, Feb 2012. <http://developer.android.com/guide/topics/security/security.html>.
19. Graphviz. Graph Visualization Software. <http://graphviz.org/>.
20. J. Heer and M. 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.
21. J. I. Hong and J. A. Landay. Webquilt: a framework for capturing and visualizing the web experience. In *Proceedings of the 10th international conference on World Wide Web*, WWW '01, pages 717–724, New York, NY, USA, 2001. ACM.
22. P. Hornyack, S. Han, J. Jung, S. Schechter, and D. 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.
23. T. Hunt. Secret iOS business; what you dont know about your apps. *Troy Hunt's Blog*, Oct 2011.
24. Ian Paul. Android Market Tops 400,000 Apps. *PCWorld*, 01 2012. http://www.pcworld.com/article/247247/android_market_tops_400000_apps.html.
25. J. Jung, A. Sheth, B. Greenstein, D. Wetherall, G. Maganis, and T. 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.
26. P. G. Kelley, J. Bresee, L. F. Cranor, and R. W. Reeder. A "nutrition label" for privacy. In *Proceedings of the 5th Symposium on Usable Privacy and Security*, SOUPS '09, pages 4:1–4:12, New York, NY, USA, 2009. ACM.
27. A. Kittur, E. H. Chi, and B. 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.
28. A. Moser, C. Kruegel, and E. Kirda. Limits of static analysis for malware detection. In *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, pages 421–430, dec. 2007.
29. K. Parrish. Storm8: We're Not Collecting Private iPhone Data. Dec 2009. <http://www.tomsguide.com/us/iPhone-Developer-Storm8-Private-Data,news-5283.html>.
30. T. Shields. Mobile Apps Invading Your Privacy. *Vercode Blog*, Apr 2011. <http://www.veracode.com/blog/2011/04/mobile-apps-invading-your-privacy/>.
31. E. Smith. iPhone Applications & Privacy Issues: An Analysis of Application Transmission of iPhone Unique Device Identifiers (UDIDs). Oct 2010. <http://www.pskl.us>.
32. T. Takala, M. Katara, and J. 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.
33. J. Tam, R. W. Reeder, and S. Schechter. I'm Allowing What? Technical Report MSR-TR-2010-54, Microsoft Research, 2010.
34. A. 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>.
35. S. Thurm and Y. I. Kane. Your Apps Are Watching You. *The Wall Street Journal*, Dec 2010.

36. WMUR. Police: Thieves Robbed Homes Based On Facebook, Social Media Sites. Sep 2010.
<http://www.wmur.com/r/24943582/detail.html>.
37. M. Wogalter. *Handbook of Warnings*. Lawrence Erlbaum Associates, Hillsdale, NJ, USA, 2006.
38. Y. Zhou, X. Zhang, X. Jiang, and V. 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.