

Capturing policies for fine-grained access-control

Prajit Kumar Das
University of Maryland,
Baltimore County
prajit1@umbc.edu

Anupam Joshi
University of Maryland,
Baltimore County
joshi@umbc.edu

Tim Finin
University of Maryland,
Baltimore County
finin@umbc.edu

ABSTRACT

In 2010, the number of mobile devices in the world surpassed the number of personal computers. Mobile devices carry confidential data, both personal and corporate. As a result, mobile devices have become a lucrative target for attackers, and privacy and security of these devices have become a vital issue since. The existing access control mechanisms in most devices, which relies on install time permission granting, is too restrictive and inadequate, since it cannot factor in the context of the device and the user. In other words, the access granted a subject can change based on the context of the device. In this paper we present Mithril, a context-driven dynamic policy-based model for defining access control on mobile devices. We describe the design of the system that captures these policy rules which are defined using Semantic Web technologies, protects a user's mobile device by executing these rules. Specifically, this paper address the question of how such policies can be obtained. We describe an iterative process that helps users in reaching their access control policies by informing them about potential policy violations on their devices. Our evaluation shows that such an iterative process is capable of capturing a specific user's policy.

Categories and Subject Descriptors

D.4.6 [Operating Systems]: Security and Protection—*Access Controls*; H.3.4 [Information Storage and Retrieval]: Systems and Software—*Current awareness systems*

Keywords

Access Control, Semantic Web, Context-Aware Computing

1. INTRODUCTION

As the number of internet connected mobile users increases, and confidential information is increasingly stored on these devices, they become attractive targets of attack. McAfee

Labs 2014 report ¹ predicted that mobile technologies would see an escalation of attacks due to openly available mobile malware source code. Another reason is the incentive – mobiles are being used as payment devices at point of Sale (e.g. Google Wallet, Apple Pay).

On the corporate end of things, incentive is also provided by Bring-Your-Own-Device (BYOD) policies that permit the same user owned device to be used both within and outside the corporate firewall, and allows applications to run on the device without restrictions. ZDNet² reported about a study conducted by Tech Pro in November 2014, which showed that 74% of participating organizations allow their employees to bring their own devices in to company premises, and that 60% of the organizations allowed employees to use personal devices to access company networks and data. Another 14% planned to allow the same within a year. The potential attack surface opened by such policies is also well recognized. A study conducted by Dimensional Research ³ reported that 67% of the respondents believed that securing corporate information was the greatest BYOD challenge.

The BYOD scenario can be used to motivate a key challenge in access control for mobile devices. Depending on the context of the usage, the access rights can change. For instance, it might be permissible to send some data over the corporate VPN, but not have it uploaded to Facebook! It might be OK to use the camera generally, but not inside the company facility. Reporting GPS locations to the platform provider (e.g. Google, Apple) might be fine in general, but not when inside a Sensitive Compartmented Information Facility (SCIF). Note that the current “permit at install” model of most mobile OSs is inadequate to this task. It also has no way to indicate if the sought for permissions are relevant to the task of the requester, and most users cannot make this judgement themselves. An example of this was observed in 2013, when it was revealed that the Federal Trade Commission had found out that an app ⁴ was collecting user location data. The app had over fifty million downloads and nearly a million ‘5-star’ ratings on the Google Android Play Store but was deceiving users by collecting their location and identity and sharing the same with third-party advertising networks. Solving the Access control on mobile devices is a therefore

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

WiSec'15 June 22-26, 2015, New York, USA

Copyright 2015 ACM X-XXXXX-XX-X/XX/XX ...\$15.00.

¹McAfee Labs: Threats report <http://goo.gl/oS8922>

²Research: 74 percent using or adopting BYOD <http://goo.gl/jldRtt>

³The impact of mobile devices on information security <http://goo.gl/p12HVU>

⁴Brightest Flashlight Free <http://goo.gl/qBSKrs>

a key problem.

Naturally, there have been multiple attempts at achieving the goal of managing access control on mobile (Android) devices. Efforts have also been made by the open source community to create systems that allow users to control permissions on the Android platform. Some of these efforts require custom ROMs or rooted devices. Most prominent examples of these efforts include the XPrivacy project (needs a rooted phone), the Privacy Guard project (available on Cyanogenmod, a custom Android ROM), the PDroid application (needs a rooted device). In 2012 Google introduced AppOps as part of Android 4.3. This feature was the first customizable permission model seen on the Android platform. AppOps made it possible to control the data flow on Android in a way that had not been possible before. Unfortunately, this option was removed with the release of Android version 4.4.2 and is only accessible on a “rooted” mobile device.

There have also been research efforts that aim to use either data tainting or policy driven approaches to address this problems. Some of these efforts include work done by Conti et. al. [3](CRePE), Enck et al. [5](TaintDroid) and Jagtap et al. [9](Preserving Privacy in Context-Aware Systems). CRePE described a system where security policy enforcement was carried out based on context of the smart phone. TaintDroid was a research effort where the data flow on an Android device was studied to figure out when sensitive data left the system via an untrusted application. The work of Jagtap et al. [9] focused on constraining data flow in a context-aware system using a policy-based framework. A related work by Ghosh et al. [6] used a similar policy driven approach to constrain application permissions based on context.

All these studies indicate that there has been a fair amount of effort focused on access-control on mobile devices. However, most of these works left out the research on how to create the policies that are to be enforced on the mobile devices. More recently efforts [1, 13, 10, 11] have been made to determine the permissions a user or a class of users might prefer. In this paper, we focus on automatically capturing a user’s access control policies that are fine-grained, dynamic and context-dependent. We also present an end-to-end system design that allows enforcement of such rules on a mobile device. In order to carry out these two tasks, we present **Mithril**⁵, a system for capturing fine-grained (context dependent) user policies and managing privacy and security on mobile devices using said policies. We adopt an iterative model for capturing specific user policy. We begin the process by using an initial default policy and observe all violations of said policy that happens on the mobile device. We use these violation data to improve the policy using user-feedback.

The rest of the paper is organized as follows. We start with a system overview in the next section. The third section provides the details of our methodology for capturing specific user policy on a mobile device. We present our experimental evaluation methods in the fourth section followed by the results of our study and discussion of our results in the fifth section. We contrast our work with the related

work found in the literature in the sixth section. Finally we conclude the paper with a summary of the current work and our future research goals.

2. SYSTEM OVERVIEW

We present the end-to-end system architecture of MITHRIL in Figure 1. MITHRIL contains four main components, i.e. policy enforcement module, policy decision module, policy store module, user policy control module. The system sits in between the Apps installed on a user’s mobile device and the Android framework. The input to the whole system is a request for data from installed apps on the mobile device. The output of the system is a response containing data or access to a component or an exception stating that the data or component is unavailable. Inside the system the data flows through the policy enforcement module to the policy decision module followed by a request to the policy store and an optional call to the user policy control depending on the request. Further details of the workings of the system’s modules is provided in the following sections.

MITHRIL, has two operating modes, i.e. OBSERVER and ENFORCER. In the observer mode the system simply stores violation of current policy. After an initial round of data collection and user interaction the system moves to the enforcer mode where it applies the current policy. It keeps on collecting data about any further violations in this mode too, to be intimated to the user in a periodic manner.

2.1 Definitions: Context, Rule, User-Category, Policy

Following are some of the important definitions that we use in this paper to describe the functionality of MITHRIL.

DEFINITION 1. *CONTEXT has been defined by Dey and Abowd [4] as: “[...] any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and application, including the user and applications themselves.” Dey and Abowd [4] also decompose context into two categories: PRIMARY CONTEXT PIECES (i.e., IDENTITY, LOCATION, ACTIVITY, TIME) and SECONDARY CONTEXT PIECES (pieces of context that are attributes of the primary context pieces for example: a user’s phone number can be obtained using the user’s identity).*

DEFINITION 2. *A RULE represented in a logic-based form states the access control ACTION that will be taken by the system given a certain user CONTEXT, a requester of data and a requested resource. The three possible actions in our rule are ALLOW, ALLOW WITH CAVEAT, DENY.*

The “caveat” refers to an option whereby the system obfuscates the data returned to the requester. For example we could obfuscate the real location of the user by sharing mock GPS coordinates [2].

DEFINITION 3. *A USER-CATEGORY is a classification of a user based on their profession.*

DEFINITION 4. *A POLICY, consists of a set of RULES (also referred to as POLICY RULES in this paper), that define access control for data. A policy is applicable to a particular USER-CATEGORY.*

⁵Mithril is a reference to a precious, lightweight and extremely strong silvery metal from the Lord of the Rings which protected its wearer, Frodo, from life threatening dangers: <http://goo.gl/sh2jXG>

2.2 Policy Enforcement

The policy enforcement module is the entry point for our system. It receives as input, data requests from apps and serves them with data as dictated by the “action” returned by the policy decision module. In the observer mode, the policy enforcement module does not control any data flow on the mobile device. In this mode it simply passes the data request tuple consisting of the requested component name or type of data and the requester name (henceforth refereed to as: request meta-data) to the policy decision module. In the enforcer mode, it passes on the request meta-data but expects the policy decision module to provide an “action”. If the action is to allow data flow, it simply makes a request to the Android framework for the data and returns the same to the requesting app. If the action is to deny the data flow, it prohibits the request from going any further. If the action is to “allow with caveat” then the data is obtained from the Android framework and a data obfuscation sub-module modifies the data before passing it on to the requesting app. Obfuscation could be done by faking location information [2] or other data from the Android framework.

2.3 Policy Decision

The policy decision module receives as input, the request meta-data from the policy enforcement module. The current context is obtained using a context synthesizer sub-module. The context synthesizer keeps updated user context facts obtained from a reasoner using ontologies to infer associations between sensor data and user context. A similar technique was explored in [7]. We use the Platys ontology [9] to semantically represent user context and app meta-data. We use classes defined in the Platys ontology to define hierarchical context models that enables us to generalize or specialize over primary and secondary user context pieces. An example of how this is used is shown in section 3.

We use a knowledge-base on the phone that stores facts about apps including app categories. The facts are extracted from various sources like the Android Marketplace ⁶ and the DBpedia ontology [12]. The facts include meta-data like app manufacturer, download count, maturity rating, user rating, developer country of origin, number and types of permissions requested by the app etc. The facts about the user context and apps are stored in form of RDF triples, which help us query the knowledge-base for properties like app types or location types. These information enables the inference mechanism as the rules are stated in terms of the type properties of apps and user context.

The final piece of information needed to make a decision are the rules for the current request meta-data, which are provided by the policy storage module. A requester, resource tuple can have multiple policy rules applicable based on contextual conditions. Once the rules are obtained, using the context and app facts from the knowledge-base a specific rule applicable is inferred by an OWL-DL reasoner. The consequent of the chosen rule is the applicable action. If the action is deny or allow with caveat, then the data request is marked as a possible violation of current policy rules.

In the observer mode, the violation meta-data, which consists of the request meta-data along-with the applicable rule

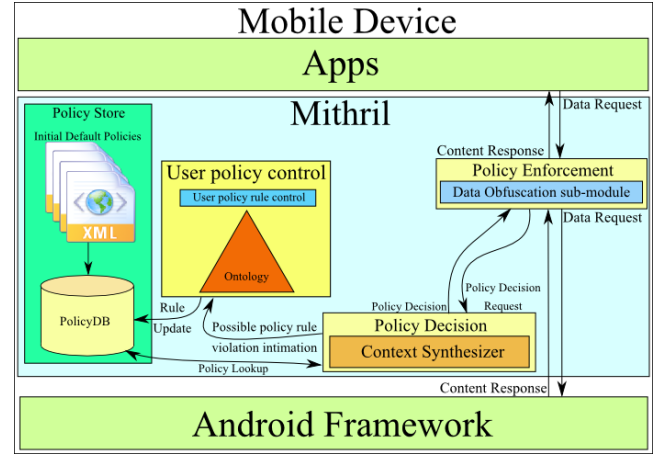


Figure 1: System Architecture

and user context is forwarded to the User Policy Control module and no response is sent to the policy enforcement module. In the enforcer mode however, the action inferred by the reasoner is returned to the policy enforcement module and at the same time the violation meta-data is forwarded to the User Policy Control module.

2.4 Policy Store

The policy storage module has a database containing the currently applicable policy for the user-category of the mobile device’s user. The user chooses an applicable user-category, when the system starts for the first time and the default for said category is then downloaded on the mobile device. The storage module receives as input a requester app information and information about the requested resource. It searches the policy database for the applicable policy rules and returns the same to the policy decision module. The second task that the policy storage handles is updating a policy rule as requested by the user policy control module. Let us take a look at how rules are represented in MITHRIL.

2.4.1 Rule Representation

Rules, in our system, are represented using the Semantic Web Rule Language (SWRL) [8]. Rules are composed of antecedents which define the context in which a certain rule is applicable, the requesting entity and the requested resource. The consequent of a rule defines the action to be taken. Following is an example rule where, we have an app that belongs to the social media category. We are taking a look at a rule from the policy for a graduate student. The rule states that while the student is in her university building, social media apps are not allowed to access the camera on her mobile device. We call this rule SOCIALMEDIACAMERAACCESSRULE and can be seen in Figure 2. The policy is called GRADSTUDENTPOLICY. Given all of above assumptions, we represent the afore-mentioned SocialMediaCameraAccessRule as:-

We can have a more detailed version of the same rule with more conditions incorporated. The resultant rule would be more complex but would give higher degree of granularity with respect to their privacy and security policy rules. The more granular second rule could be stated as “do not allow camera access to SocialMedia apps when the time of day is between 9AM and 5PM and it is a weekday and the user is

⁶Android Market Place: <http://goo.gl/4GHoFo>

```

resourceRequested(?r, Camera) ∧
requestingApp(?app) ∧
hasAppType(?app, SocialMedia) ∧
User(?u) ∧
userLocation(?u, ?l) ∧
hasLocationType(?l, UniversityBuilding)
→
AccessLevel(Deny)

```

Figure 2: Simple rule for controlling social media camera access

```

resourceRequested(?r, Camera) ∧
requestingApp(?app) ∧
hasAppType(?app, SocialMedia) ∧
User(?u) ∧
userTime(?u, ?t) ∧
timeAfter(?t, 0900) ∧
timeBefore(?t, 1700) ∧
userDayOfWeek(?u, ?d) ∧
hasDayType(?d, weekday) ∧
userActivity(?a) ∧
hasActivityType(?a, Advisor_Meeting) ∧
userpresenceInfo(?p) ∧
hasPresenceType(?p, Advisor) ∧
userLocation(?u, ?l) ∧
hasLocationType(?l, UniversityBuilding)
→
AccessLevel(Deny)

```

Figure 3: Rule with higher granularity, for controlling social media camera access

at university building location in presence of his advisor and has a meeting scheduled with her advisor”. We can see this rule in Figure 3.

3. USER POLICY CONTROL

Mithril uses user feedback to iteratively modify rules on the mobile device. A feedback iteration starts with a list of violations, obtained from the policy decision module, being presented to the user. When the user chooses to look at a specific rule violation from the list they are presented with the specific rule’s violation meta-data, as seen in Figure 4. The violation meta-data include the actual rule statement and a list of facts about the app that is violating the rule. The user then has the option of further exploring the violation by clicking on the “Display Policy Rule Conditions” button for the context antecedents for the rule.

In each iteration we show to the user the potential violations that have been captured on the mobile device. The user has two options at this point. They can choose to state a violation as a true violation or as a false violation. If they denote a violation as false, we request them to further provide feedback about what should be the modification in the policy rule. As described in the section 2.3, our ontology and user context facts allows us to generalize or specialize over user’s context. This provides a convenient way for the user to modify the policy conditions, in order to define the changes in the current rules. Let us consider an example to understand the mechanism better. Referring to the policy presented in Figure 2, we assume that we have the user at

User Policy Rule Control	
Static Information	
Policy Rule Information	
Policy Name: GradStudentPolicy	
Rule Name: SocialMediaCameraAccessRule	
Requester Information	
App Name: Instagram	
App Content Maturity Rating: Medium	
App Developer Name: Instagram	
App Developer Origin Country: USA	
App Rating: 4.5	
App Installation Count: 1-5 million	
Violations Information	
Access allowed to: Camera	
Contextual Violation Aspect: Policy rule was to <i>deny camera access</i> , at <i>university building</i> for <i>social media apps</i> .	
Dynamic Policy Rule Conditions	
Delete Rule	Save Rule
Create New Rule	

Figure 4: Rule violation meta-data displayed to user

a location ‘CS Building, NYU, NY, USA’ which is a University Building as per our ontology. Our ontology allows us to generalize the policy condition for location to: ‘NYU, NY, USA’ which is a University Campus or specialize it to: ‘Lab 1234, NYU, NY, USA’ which is a University Lab. On choosing to modify a specific rule, the options that are visible to the user are based on such a hierarchical context model. A sample view of the hierarchical choices can be seen in Figure 5. A modification to a rule can therefore be carried out, in the following ways defined below:-

- A policy rule’s consequent could be modified
- One or more antecedent(s) could be modified
- One or more contextual antecedent(s) could be added to the list of antecedents currently applicable
- One or more of the currently applicable antecedent(s) could be deleted
- A policy rule could be deleted completely
- A new policy rule could be added to the policy set

Policy rules in MITHRIL are defined in a generic form. Take a look at the rule in Figure 6. Here the rule is applicable at a work location. Our ontology allows us to semantically define a user’s context and therefore we are able to

Dynamic Policy Rule Conditions		
Time period related conditions	Everyday	9:00 AM
Condition not applicable at the moment Click here to enable	Weekday	To
	Weekend	
	Monday	
		5:00 PM
Location related conditions	Country City/State University Campus University Building University Lab	
Activity related conditions	Public Meeting Department Colloquium Research Group Meeting Advisor Meeting	
Condition not applicable at the moment Click here to enable		
Presence of individual related conditions	Academicians Professors Advisor	
Condition not applicable at the moment Click here to enable		
Add additional conditions	Environment Conditions Activity Conditions Presence Conditions	

Figure 5: Ontology-driven hierarchical options for rule modification

```

resourceRequested(?r, Camera) ∧
requestingApp(?app) ∧
hasAppType(?app, SocialMedia) ∧
User(?u) ∧
userLocation(?u, ?l) ∧
hasLocationType(?l, Work)
→
AccessLevel(Deny)

```

Figure 6: Simple rule for controlling social media camera access at generic location context

infer that for a graduate student a work location is a Lab or University location. However, what happens if our user is visiting another lab or university to meet friends? Our policy would naturally ensure that MITHRIL will assume that the camera access needs to be blocked. In this case a rule that is generic needs to be modified. The way we handle this is, the user has the option of disabling a rule or a complete policy when needed by explicitly issuing such an instruction. However, we collect the violation meta-data and store it for the next iteration of user policy control feedback mechanism.

In both modes of operation for MITHRIL, the user policy control module receives violation information. It records these to be shown to the user at a later stage. The frequency at which a user will be asked to edit their policy rules has been left as a user prerogative for now. In each iteration we record statistics of changes happening on the device and use to compute our distance from an ideal goal.

It is clearly observable that our policy rules are signifi-

cantly more complicated as opposed to a simple permission based model that Android follows by default. The dynamic nature allowed by the variable actions and the granularity provided by the contextual antecedents are contributing factors to this complexity. However, it also gives more control to the user over her data. In our research we show that it is possible to start from an generic policy applicable to a class of users and reach a state where we have captured specific policies for a user from that category. We show the same through our experiments explained in the following section.

4. EXPERIMENTAL EVALUATION

As the main focus of this paper is to capture specific user policies, we focus our evaluation on the same. Following are the ways we have evaluated our system. We installed MITHRIL on several user devices and on each of those devices we started with an initial default policy. For the sake of simplicity we will denote all default policies with the letter P. For the purposes of our experiment we requested ten graduate students to provide us their feedback. Before we ask them to use our system, we also ask them to use a web app to modify the default graduate student policy to specify their own specific policy.

As stated before, MITHRIL has two operating modes, observer and enforcer. In the observer mode the policy P is taken as a reference point and app activity on the mobile device is monitored. Any violation, as detected by the policy decision module is recorded and then at a pre-defined time period, the user is presented with these potential violations. The user then modifies the rules, if necessary. In our experiments we record on a per iteration basis, the following statistics:-

- Number of rule violations recorded.
- Number of rule changes made by user.
- Number of condition changes made per rule.
- Distance from their ideal policy.

Through our evaluations we are trying to find out the fraction of the users who are able to reach their ideal policy as defined at the start of the experimental evaluation.

4.1 Data collection

5. RESULTS

6. RELATED WORK

Research being done to predict user’s preferences by a number of people [1, 13, 10, 11]. Owing to that research we make an assumption that it is possible to fairly accurately create user permission choices on Android devices. However, our goal is separate from theirs in a threefold manner. Firstly we are defining policy rules for users which may allow, deny or allow with caveat specific permissions depending on the user context. Secondly we are not trying to show that it is possible to learn a user’s policy from scratch rather we are agreeing with their observation that it is possible to use privacy profiles to define or group user preferences [11]. Instead we are trying to show that with user feedback it is possible to reach an individual user’s “perfect” policy with a

certain probability. Thirdly, we are researching ways to include app provenance information, api usage and observed mobile behavior [5] to compute a metric that will accurately measure the trustworthiness of an app.

7. CONCLUSIONS

8. ACKNOWLEDGMENTS

Support for this work was provided by NSF grant 0910838, MURI award FA9550-08-1-0265 from the Air Force Office of Scientific Research.

9. REFERENCES

- [1] M. Benisch, P. G. Kelley, N. Sadeh, and L. F. Cranor. Capturing location-privacy preferences: Quantifying accuracy and user-burden tradeoffs. *Personal Ubiquitous Comput.*, 15(7):679–694, Oct. 2011.
- [2] A. R. Beresford, A. Rice, N. Skehin, and R. Sohan. Mockdroid: Trading privacy for application functionality on smartphones. In *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications*, HotMobile '11, pages 49–54, New York, NY, USA, 2011. ACM.
- [3] M. Conti, V. T. N. Nguyen, and B. Crispo. Crepe: Context-related policy enforcement for android. In M. Burmester, G. Tsudik, S. Magliveras, and I. Ilic, editors, *Information Security*, volume 6531 of *Lecture Notes in Computer Science*, pages 331–345. Springer Berlin Heidelberg, 2011.
- [4] A. K. Dey and G. D. Abowd. Towards a better understanding of context and context-awareness. In *First Int. symposium on Handheld and Ubiquitous Computing (HUC)*, 1999.
- [5] 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*, pages 1–6, 2010.
- [6] D. Ghosh, A. Joshi, T. Finin, and P. Jagtap. Privacy control in smart phones using semantically rich reasoning and context modeling. In *Security and Privacy Workshops (SPW), 2012 IEEE Symposium on*, pages 82–85, 2012.
- [7] T. Gu, X. H. Wang, H. K. Pung, and D. Q. Zhang. An ontology-based context model in intelligent environments. In *Communication Networks and Distributed Systems Modeling and Simulation Conf. (CNDs)*, 2004.
- [8] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Groszof, M. Dean, et al. Swrl: A semantic web rule language combining owl and ruleml. *W3C Member submission*, 21:79, 2004.
- [9] P. Jagtap, A. Joshi, T. Finin, and L. Zavala. Preserving privacy in context-aware systems. In *Fifth IEEE Int. Conf. on Semantic Computing (ICSC)*, 2011.
- [10] J. Lin, B. Liu, N. Sadeh, and J. I. Hong. Modeling users' mobile app privacy preferences: Restoring usability in a sea of permission settings. In *Symposium On Usable Privacy and Security (SOUPS 2014)*, pages 199–212, Menlo Park, CA, July 2014. USENIX Association.
- [11] B. Liu, J. Lin, and N. Sadeh. Reconciling mobile app privacy and usability on smartphones: Could user privacy profiles help? In *Proceedings of the 23rd International Conference on World Wide Web, WWW '14*, pages 201–212, New York, NY, USA, 2014. ACM.
- [12] P. N. Mendes, M. Jakob, and C. Bizer. Dbpedia: A multilingual cross-domain knowledge base. In *LREC*, pages 1813–1817, 2012.
- [13] N. Sadeh, J. Hong, L. Cranor, I. Fette, P. Kelley, M. Prabaker, and J. Rao. Understanding and capturing people's privacy policies in a mobile social networking application. *Personal Ubiquitous Comput.*, 13(6):401–412, Aug. 2009.