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

Smart phones have become ubiquitous due to its low cost and Android is the biggest player in the market. It boast of more than a billion active 30 day users according Engadget $\mathrm{Engadget}_{m}arket_{s}hare, which comes down to 84.8\% market share in the s$ $store s co - exist which include Amazon App Store, Slide Me, 1 Mobile Mo$

The proliferation of smart phones encouraged all the domains to be linked to the smart phone for easier and faster access. Medical domain is also not different and a number of hardware accessories had come up. Some of them include Blood glucose monitoring system[5], Blood pressure monitor[6], Heart monitor[7], Fitness accessories like Fitbit, Breathalyzer[4] etc. All these accessories have an associated Android application which interacts with the physical device, process the data, display it and stores relevant information. Another category of applications are those used by hospitals and its employees to manage information about the patients, doctors etc. The important point is that all these sensitive information are being received, processed, stored and possibly updated to cloud storage by these applications. The fact that personal and sensitive data is handled by these devices presents a potential security risk. With more than 1.6 million apps to choose from, let alone a layman user, even a computer expert will have difficulty to understand what are benign and what are malicious. One of the most traditional ways to look at this problem is to use signature based malware detection, in which once a malware application is detected, signature will be created once and the signatures are updated to a common database. Any future appearance of the same malware would be detected using these signatures. The classical problem with this approach is the occurrence of mutating malwares which changes itâĂŹs signature every time it is spread and it inability to detect future similar attacks. Apart from this Android apps has issues on its own. One of the main problem is the high volatility of the app markets. Since unlike Apple store, Google play store is not verified and hence new applications becomes available and taken out quite frequently. Yet another issue is repackaging popular application with malware. Often these repackaged applications are uploaded to other local App-stores where the original application is not available. Hence we require a faster and dynamic setup to detect unsafe or rather potentially unsafe applications. [Paper to be referred Dr B]Even though not totally accurate, there are some intuitions which we can utilize from the meta-data available with each application. The different meta-data available with each of these applications include App description, Permissions required, Number of downloads, Broad Application categories etc. As an example of our intuition let us take the case of âĂIJbrightestlightâĂİ application, which is available in Google Play Store. The description of the application says that it is a flash light app, which shows some unobtrusive

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

Ads. But on analyzing the Permissions required, we can see that the application requests permissions for location (accurate GPS and approximate network based), file manipulation (read, modify and delete USB storage), camera (curiously it includes auto-focus permission) etc. This does seem suspicious. But still we cannot say that the application is unsafe, rather we need further investigation.

2. RELATED WORK

Playdrone : Crawls Playstore- how playstore evolved - source code analysis of library usage - similar app detection-secret authentication key storage (can be found by decompilation) (1) native libraries are heavily used by popular Android applications, limiting the bene

ts of Java portability and the ability of Android server ooading systems to run these applications, (2) 25of Google Play is duplicative application content, and (3) Android applications contain thousands of leaked secret au-thentication keys which

Andradar : First, we can discover malicious applications in alternative markets, second, we can expose app distribution strategies used by malware developers, and third, we can monitor how different markets react to new malware. to identify and track malicious apps still available in a number of alternative app markets.

Android Security : discuss the Android security enforcement mechanisms, threats to the existing security enforcements and related issues, malware growth timeline between 2010 and 2014, and stealth techniques employed by the malware authors, in addition to the existing detection methods. This review gives an insight into the strengths and shortcomings of the known research methodologies and provides a platform, to the researchers and practitioners, toward proposing the next-generation Android security, analysis, and malware detection techniques.

ANDRUBIS:, a fully automated, publicly available and comprehensive analysis system for Android apps. ANDRUBIS combines static analysis with dynamic analysis on both Dalvik VM and system level, as well as several stimulation techniques to increase code coverage.

changes in the malware threat landscape and trends amongst goodware developers. Dynamic code loading, previously used as an indicator for malicious behavior, is especially gaining popularity amongst goodware

App analysis for astma!!

App behavoir against description CHABADA tool clustering apps by description topics, and identifying outliers by API usage within each cluster, our CHABADA approach effectively identifies applications whose behavior would be unexpected given their description. Recommendations for android eco system

3. SYSTEM OVERVIEW

We present the end-to-end system architecture of MITHRIL in Figure **??**. MITRHIL 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.

MITRHIL, 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.

3.1 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 [**?**] 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 [**?**] 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 [**?**].

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.

3.2 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

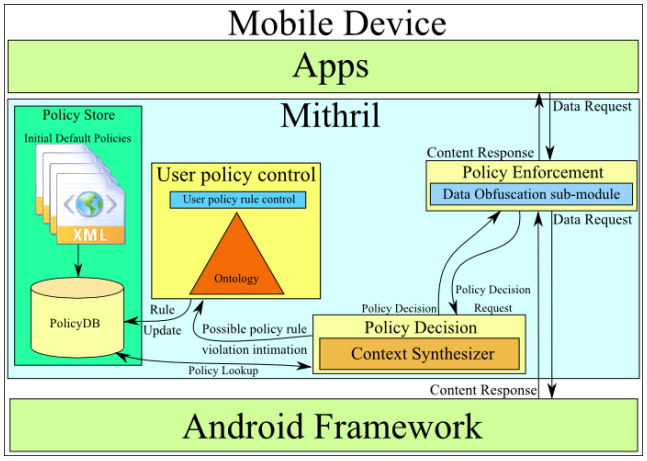

Figure 1: System Architecture

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 [**?**] or other data from the Android framework.

3.3 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 for context inference from low level sensor information was explored in [**?**]. We use the Platys ontology [**?**] 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 **??**.

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 [1] and the DBpedia ontology [**?**]. 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.

[1]Android Market Place: `http://goo.gl/4GHoFo`

$$
\begin{aligned}
&resourceRequested\,(?r, Camera) \wedge \\
&requestingApp\,(?app) \wedge \\
&hasAppType\,(?app, SocialMedia) \wedge \\
&User\,(?u) \wedge \\
&userLocation\,(?u, ?l) \wedge \\
&hasLocationType\,(?l, UniversityBuilding) \\
&\rightarrow \\
&AccessLevel\,(Deny)
\end{aligned}
$$

Figure 2: Simple rule for controlling social media camera access

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 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.

3.4 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.

3.4.1 Rule Representation

Rules, in our system, are represented using the Semantic Web Rule Language (SWRL) [**?**]. 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 **??**. 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 at university building location in presence of his advisor and

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

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

has a meeting scheduled with her advisor". We can see this rule in Figure **??**.

4. 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 **??**. 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 **??**, 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 **??**, we assume that we have the user at 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 **??**. 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

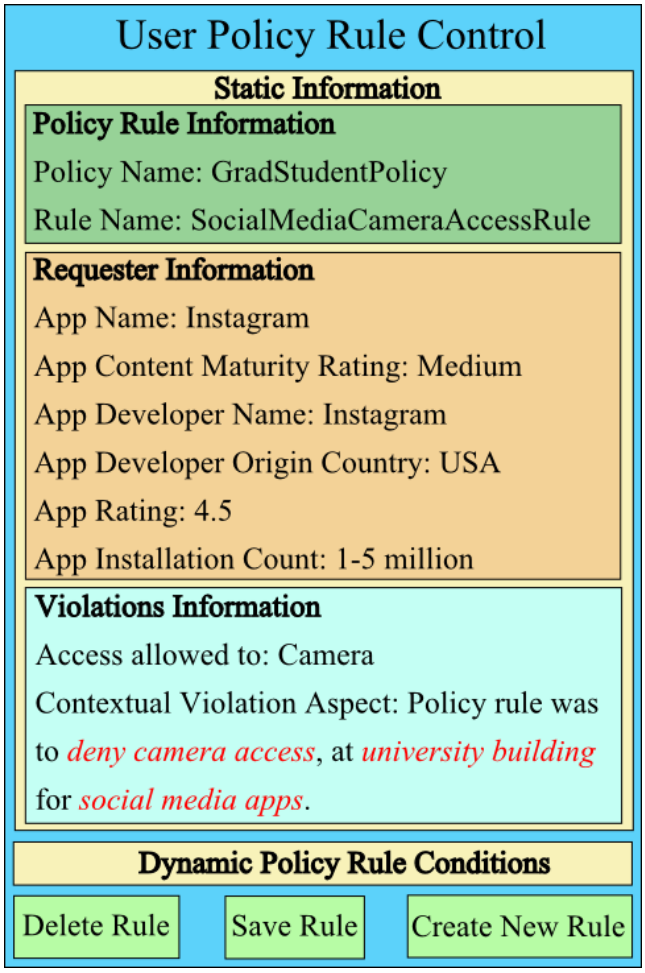

Figure 4: Rule violation meta-data displayed to user

- 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 **??**. 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 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

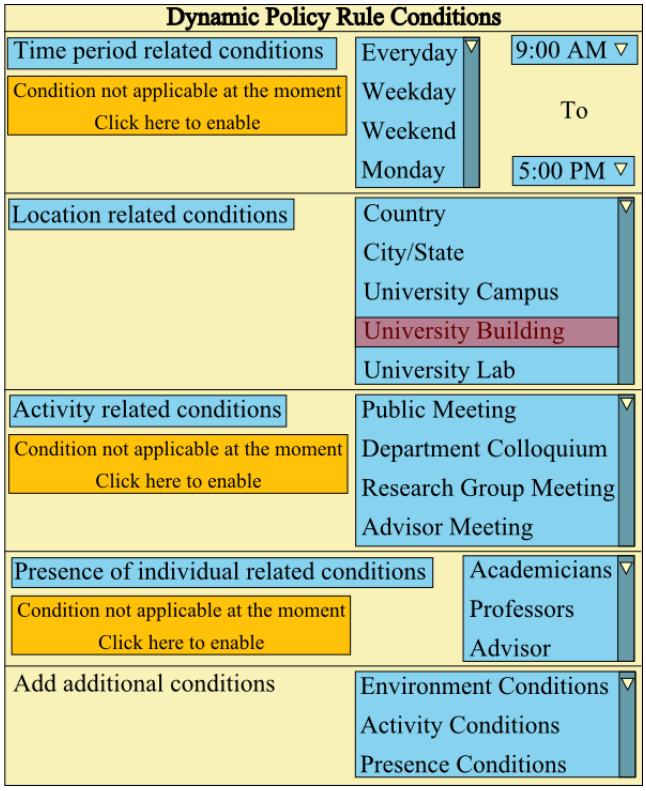

Figure 5: Ontology-driven hierarchical options for rule modification

$$
\begin{aligned}
&resourceRequested\,(?r, Camera) \wedge \\
&requestingApp\,(?app) \wedge \\
&hasAppType\,(?app, SocialMedia) \wedge \\
&User\,(?u) \wedge \\
&userLocation\,(?u, ?l) \wedge \\
&hasLocationType\,(?l, Work) \\
&\rightarrow \\
&AccessLevel\,(Deny)
\end{aligned}
$$

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

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 significantly 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.

5. 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 MITRHIL 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, MITRHIL 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.

5.1 Data collection

6. RESULTS

7. RELATED WORK

Research being done to predict user's preferences by a number of people [**?**, **?**, **?**, **?**]. 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 [**?**]. 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 [**?**] to compute a metric that will accurately measure the trustworthiness of an app.

8. CONCLUSIONS

9. 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.

10. 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] 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.

[4] 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.

[5] 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.

[6] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean, et al. Swrl: A semantic web rule language combining owl and ruleml. *W3C Member submission*, 21:79, 2004.

[7] 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.

[8] 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.

[9] 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.

[10] P. N. Mendes, M. Jakob, and C. Bizer. Dbpedia: A multilingual cross-domain knowledge base. In *LREC*, pages 1813–1817, 2012.

[11] 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.

@online Engadget$_m arket_s hare, author = Christopher Trout, title =$
$Android still the dominant mobile OS with 1 billion active users, year =$
$2014, url = http://www.engadget.com/2014/06/25/google-io-2014-by-the-numbers/, urldate =$
$2014-06-25$@$online Online_{App} Stores, author = Ryan Chang, title = 10 Alternative Android App Stores, year = 2014, url = http://$