Personalizing context-aware access control on mobile platforms

Prajit Kumar Das, Anupam Joshi and Tim Finin
Department of Computer Science and Electrical Engineering
University of Maryland, Baltimore County
{prajit1, joshi, finin}@umbc.edu

***Abstract*—Context-sensitive access control has been a research topic within mobile computing for more than a decade. Much of the work has focused on modeling context and representing policies. Choosing an appropriate policy for a user, however, remains a challenging goal. Creating usable mobile access control solutions have been researched from a users' permission control perspective. We present a study carried out with subjects using their personal mobile devices that captures individualized policies through an iterative user feedback process. Policy precision, also referred to as "Violation Metric" (VM), was used to decide when all necessary policies had been captured. The feedback process used a hierarchical context ontology to represent user-context and gathered contextual-situations in which a policy would be applicable. The study also investigated the feasibility of using the VM measure to determine completion of the capture process for the users' personalized access control policies, that handles their mobile privacy and security needs. Using an appropriate pre-defined policy is shown to have lesser user impact when trying to personalize access control policies for users.**

I. Introduction

Mobile platforms were predicted to experience an escalation of attacks by a 2014 McAfee Threats Report [1] due to openly available mobile malicious source code. The spate of recent mobile malware discoveries indicate that the prediction is turning into a reality and leading mobile platform providers are constantly fighting against such attacks [2]. Mobile malware is not the only threat faced by users, however. A Google taxonomy provides us with additional threats through Potentially Harmful Apps (PHA) [3]. These threats include Billing Frauds, Spyware, Hostile Downloaders, Privilege Escalators, Ransomware and Rooting Apps.

To protect against PHA threats, mobile platforms use certain typical security features on user devices. In addition to application scanning by market places, these security features include the usage of application sandboxes for mobile apps operations and a permission based security model to provide access to system resources. Users are expected to decide which of these access requests are to be allowed and which ones should be denied. Unfortunately, it has been observed that users do away with access control policy choices if there is a perceived benefit to using an app [4]. In addition, access control needs are not always a simple yes or no decision. The current "permit once" permission model implemented by most mobile operating systems is inadequate when it comes to protecting user data in presence of apps that are intent upon collecting and tracking users contextual data like location, audio, environmental information, and nearby presence data. Such data collection is common today due to the PHA threats mentioned above. In light of such potential threats, we submit that there is a need to create fine-grained, dynamic and context-driven access control policies to protect the privacy and security of a user and her data.

In this paper, we focus on developing a methodology of personalizing a set of policy rules using an approach based on a human-in-the-loop collaborative editing of an initial default policy. The goal is to create a personalized user policy that protects a user and her data, starting from the initial default policy. It is possible that one may use a trusted entity or security domain expert to create a set of policies for a known target like corporate data. However, protecting users against attacks from user-space apps requires capturing a user's personal preference with respect to such apps.

For this purpose, we have created the MITHRIL [1] framework [5]. MITHRIL is an end-to-end context-dependent access control framework that monitors the activities of applications on a user's mobile device in various contextual situations and captures their access control preferences in that context. We use a *violation metric* as the theoretical model for our framework. It helps us to determine convergence of the policy capture process. MITHRIL consists of two components: MithrilAC and a back-end app analytics module. In this paper, we describe the MithrilAC component, discuss its implementation, and outline its evaluation through a user study to show the feasibility of using a precision metric over the captured policy to determine when the adaption process is complete.

This paper addresses the following research question. **RQ1:** Given an initial policy P and user's implicit desired goal policy P', can the violation metric be used to determine the completion of the capture process?

MithrilAC is a component of the MITHRIL framework and its mobile access control middleware. In MithrilAC [2] we capture mobile application behavior along-with user-context and compare them to currently known policy. Any deviation from the known policy is then submitted for review by the

[1]MITHRIL is a precious, lightweight and extremely strong silver steel from the Lord of the Rings which protected its wearer, Frodo, from life threatening dangers

[2]MithrilAC requires certain operating system level privileges. Android being open source allows us to make these changes so we have used it for our prototype building but the concepts we have used applies to all mobile platforms.

user and her feedback helps MITHRIL refine the policy, thus capturing the user's access control needs. The refinement process is complete when no new deviations are observed or the precision of the captured policy is above a predefined threshold. We use the Semantic Web Rule Language [6] (SWRL), to represent our access control policy rules. We use the Platys ontology [7] that is represented using the Web Ontology Language (OWL) [8] to model a hierarchical notion of user context. MITHRIL combines information about users' context, requested information and requester information as antecedents in policy rules that allows us to express complex rule conditions. We also show that using a curated initial default policy, instead of a default deny policy, leads to a reduction in user interactions required in the feedback process.

The main contribution of our work is the design and development of the MithrilAC system that achieves the following four objectives:

- Create a mobile-middleware that uses human-in-the-loop collaborative editing of access control policies.
- Implement a middleware that displayed run time app activity to user in order to assist them in edit their policy preferences.
- Show the feasibility of using "Violation Metric" as a way to determine completion of policy capture process through a user study.
- Show that a curated initial default policy reduces the amount of user interaction required.

The rest of the paper is organized as follows. We start with a discussion of the related work in Section II. Following that, we present our system's overview in Section III. We describe the user policy control process in Section IV. We present our evaluation methodology and results in Section V followed by a discussion about the user study results in Section VI. Finally, we conclude the paper with a summary and discussion on possible future directions for the work in Section VII.

II. Related Work

The domain of access control is well researched. Role Based Access Control (RBAC) [9] and Attribute Based Access Control (ABAC) [10] are two popular models that have been used for managing access control in various domains. In the mobile domain, Ghosh et. al. [11] used a semantically rich context model to manage data flow among applications and filter them at a deeper granularity than it was possible using available security mechanisms on smart phones. Kagal et. al. [12] used distributed policy management as an alternative to traditional authentication and access control schemes. Rei, a policy language described in OWL and modeled on deontic concepts of permissions, prohibitions, obligations and dispensations [12], used Semantic Web technologies to express what an entity can/cannot do and what it should/should not do.

In our work, we use Semantic Web technologies like a hierarchical context ontology defined using OWL. Our rules are defined using the Semantic Web Rule Language that allows us to express rules that are more expressive than ones that can be defined using OWL. We use ABAC as our access control model. The MithrilAC mobile middleware connects context data to a high-level abstraction of context and executes defined rules to protect user data.

The state-of-the-art in research on policy capture stops at determining generalized privacy profiles [13], [14], [15], [16]. These works concluded that it was possible to create privacy "profiles" applicable to user categories on mobile devices with reasonable accuracy. When it comes to defining their own rules, Sadeh et.al. [14] observed that users were not good judges of how well a rule meets their true needs or preferences. However, in other work, Sadeh et.al. [17] showed that with enough "privacy nudges" that explained how their location was being shared, users could be guided to modify their preferences. We argue that given a set of policy violations and a hierarchical context model, users are able to define their preferred policy. We focus on using context generalization and specialization with assistance from our Platys ontology [7] driven context model, combining that with user feedback to reach an individual user's preferred specific policy.

Context discovery on mobile devices [18] has shown significant success with semantic location [19], [18], activity recognition and complex activity recognition [20], [21], [22]. This work takes advantage of knowledge of context discovery techniques from earlier projects and adds a preliminary presence context detection mechanism. In addition to the standard context information of location and activity, we use Android Nearby APIs and Bluetooth IDs to discover presence information. In our previous work [23], we used the Nearby API [24] from Android along with beacons to answer questions while preserving privacy of the enterprise data. We incorporate those techniques in the current work but reverse it to detect the presence of actors relevant to our user.

III. System Overview

In this section, we describe the system architecture for MithrilAC mobile middleware system and how it enables the capture of user personalized access control policy. MithrilAC follows the Attribute-Based Access Control standard [25] as shown in Figure 1. The important subcomponents of MithrilAC are the Policy Decision, Policy Enforcement and User Policy Control modules. The Policy Decision module uses a context synthesizer sub-module that maintains a knowledge graph of facts about the user context. It uses an OWL-DL reasoner to infer additional relations, resulting in high-level and semantically rich context. Rules for a access request are provided by the policy storage module. Relevant policy rules are selected using a tuple composed of a requester and a resource and then filtered based on contextual conditions. Once rules are obtained, using context and application facts from the knowledge graph, a specific rule applicable is inferred by an OWL-DL reasoner. The *c*onsequent of a chosen rule is the applicable action. If action is deny, then a data request is marked as a possible violation of current policy rules. The Policy Enforcement module receives data requests from apps and serves them with data as dictated by the "action" returned by policy decision module.

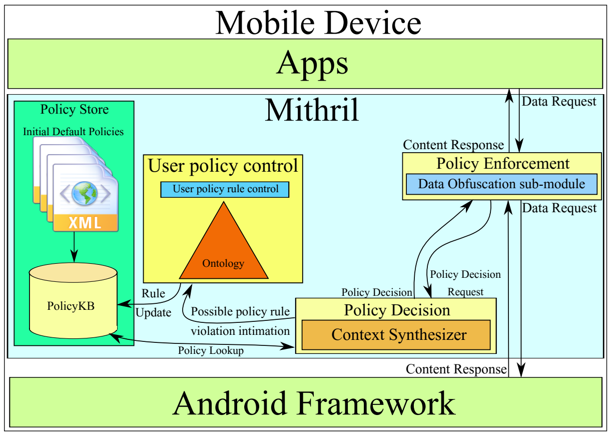


Fig. 1. MithrilAC middleware architecture

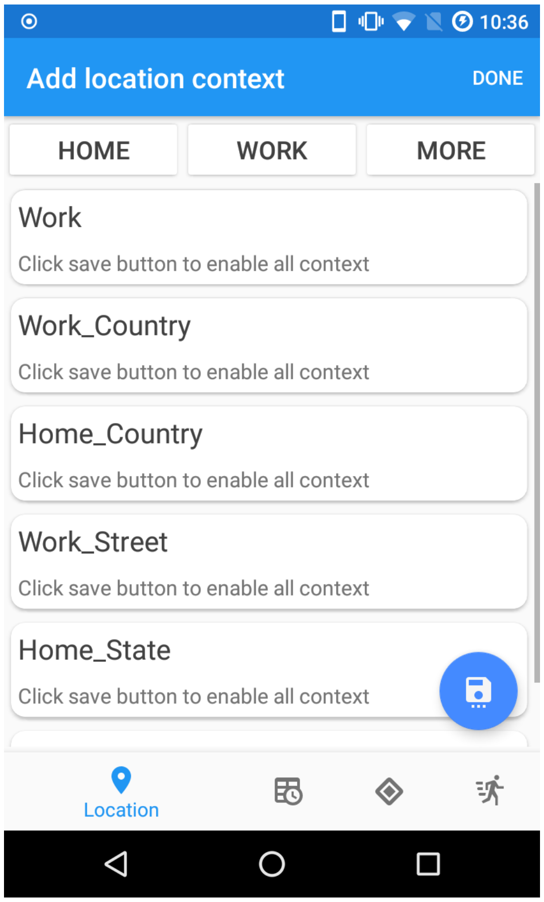
MithrilAC has administrative privileges through our custom-built ROM and uses context-dependent policy rules to manage access control on the device. There are two operational modes for the middleware; OBSERVER and ENFORCER. OBSERVER mode corresponds to the phase where the system passively observes events on a mobile device. ENFORCER mode refers to the phase when our middleware actively blocks operations on the device that are in violation of captured access control policy rules.

In observer mode, the policy enforcement module does not enforce access control on the mobile device. It simply passes data request tuples consisting of a requested component name or type of data and a requester name (henceforth referred to as *request metadata*) to the policy decision module. In enforcer mode, it passes on a request metadata but expects the policy decision module to provide an "action". If the action is to allow access, it simply makes a request to the Android framework for the data and returns the same to the requesting app. If action is to deny access, it prohibits the request from going any further. The User Policy Control module is of key importance in this paper and will be discussed in detail in Section IV, but we provide a brief overview here. As we have explained before, MithrilAC starts with an initial policy. We collect basic profile information that includes work location, home location, supervisor, family, colleague information and other facts (See Figure 2) as part of our context instantiation process. Using the policy control module, we capture a user's preferred policy. We use an ontology to define contextual information using a hierarchical context model. We use Location and Activity generalization as described in [18] and discussed further in Section IV.

A. Relevant Term Definitions

The following are some terms and definitions that we use in this paper. In our work, a policy applies to a user and her device and consists of a set of rules that control the behavior of an application in a given context. Following, is a formal definition of "rule" by Fuernkranz [26]:

Definition 1: [...] an expression of the form:



Work
Click save button to enable all context

Work_Country
Click save button to enable all context

Home_Country
Click save button to enable all context

Work_Street
Click save button to enable all context

Home_State
Click save button to enable all context

Fig. 2. User context instantiation

IF Conditions THEN c

where "c" is a class label, and "Conditions" are a conjunction of simple logical tests describing properties that have to be satisfied for the rule to 'fire'.

Definition 2: CONTEXT has been defined by Dey and Abowd [27] as: *"[...] any information that can be used to characterize the situation of an entity (i.e., identity, location, activity, time). 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."*

We have used above-mentioned definition of context and extended it further in our previous work [11] to generalize or specialize location and activity context.

Definition 3: 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 for a USER-CATEGORY or specific user.

Definition 4: A RULE, also called policy rule in our system, is a Semantic Web Rule Language (SWRL) [6] rule represented as `antecedent ⇒ consequent`.

MithilAC uses context-sensitive access control policy rules represented in SWRL to handle access control on mobile devices. Sharma et.al. [28] created a system that showed how to represent security rules using OWL and the ABAC model.

SWRL was a proposal for a rule extension for OWL (W3C member submission) and therefore we use it for our access control policy representation. The abstract syntax for SWRL rules follow the Extended Backus-Naur Form notation which, while useful for XML and RDF serializations, isn't particularly easy to read. For readability, we use the following informal format: `antecedent ⇒ consequent`. Antecedent(s) must hold for a consequent to apply. Multiple antecedents in a rule are defined as a conjunctions of atoms. The consequent atom states whether the access is allowed or denied. Antecedents in our rule specification consist of the *context of a requesting entity* along with the *entity type* that is being requested. A more abstract representation may be considered as a triple (R, C, Q) which contains: R, that represents the requester's context, C is user's context and Q is the query received by the system. Context in MITHRIL is defined using a hierarchical context model conceptualized in the Platys project [7]. The Platys ontology (see Figure 3) allows us to define a high-level abstraction of context. MithrilAC uses Semantic Web technologies to specify high-level, declarative policies in the form of SWRL rules. In this ontology, context is modeled to include a semantic notion of a Place.

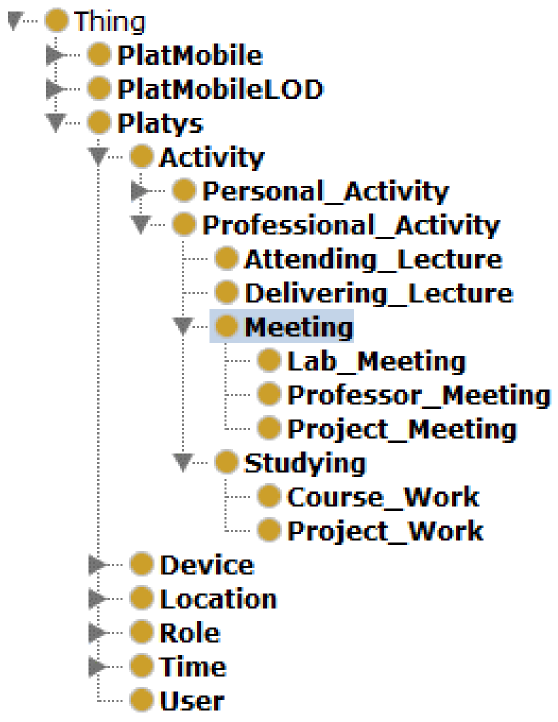


Fig. 3. Snapshot of Platys Ontology defining context hierarchy

Rule examples: Let us take a look at a rule from our policy called GRADSTUDENTPOLICY for graduate students, called SOCIALMEDIACAMERAACCESSRULE. The rule states that, while the student is in a university building, social media apps are not allowed to access the camera on her mobile device. The rule is shown Figure 4.

Example of a higher granularity rule can be seen in Figure 5,

```
@prefix platys:<https://www.ebiquity.org/ontologies
/platys/1.0>.
platys:ResourceRequested(?r, "Camera") ∧
platys:requestingApp(?app) ∧
platys:hasAppType(?app, "SocialMedia")∧
platys:User(?u) ∧
platys:userLocation(?u, ?l) ∧
platys:hasLocationType(?l, "University Lab")
⟹
platys:denyAccess("Camera")
```

Fig. 4. This simple rule for controlling social media camera access specifies that while the student is in a university building, social media apps are not allowed to access the camera on her mobile device.

which incorporates additional conditions. In plain terms, we are now stating that the device should not allow camera access to "Social Media" apps when the time of day is between 9:00am and 5:00pm and it is a weekday and the user is at a university lab location in th presence of her advisor and has a meeting scheduled with her advisor".

```
@prefix platys:<https://www.ebiquity.org/ontologies
/platys/1.0>.
platys:ResourceRequested(?r, "Camera") ∧
platys:requestingApp(?app) ∧
platys:hasAppType(?app, "SocialMedia")∧
platys:User(?u) ∧
platys:userTime(?u, ?t) ∧
platys:timeAfter(?t, "0900") ∧
platys:timeBefore(?t, "1700") ∧
platys:userDayOfWeek(?u, ?d) ∧
platys:hasDayType(?d, "Weekday") ∧
platys:userActivity(?a) ∧
platys:hasActivityType(?a, "Advisor\_Meeting") ∧
platys:userpresenceInfo(?p) ∧
platys:hasPresenceType(?p, "Advisor")∧
platys:userLocation(?u, ?l) ∧
platys:hasLocationType(?l, "University Lab")
⟹
platys:denyAccess("Camera")
```

Fig. 5. This policy rule for controlling social media camera access applies in a more specific context.

IV. USER POLICY CONTROL

Although Android APIs capture a user's location at the level of position, i.e., geospatial (latitude-longitude) coordinates, it can then be mapped to a Place or geographic entity, such as a region, political division, populated place, locality, and physical feature. A position while being valuable on its own, from the standpoint of context, Place is a more inclusive and a higher-level abstraction. Using the Platys ontology a *User* is associated with a *Device* whose *Position* maps to a geographic place (*GeoPlace*) such as *"UMBC"* and to a conceptual place (*Place*) such as *"At Work"*. Some *GeoPlaces* are part of others due to spatial containment and such relationship (*part_of*) is transitive. The mapping from *Positions* to *GeoPlaces* is many to one and the mapping from *Positions* to *Places* is many-to-many (the same *Position* may map to multiple *Places*, even for the same *User*; and, many *Positions* map to the same *Place*). Mapping from *Positions* to *Places* is done through *GeoPlaces* (*maps_to* is a transitive property). An *Activity* involves *Users* under certain *Roles*, and occurs at a given *Place* and *Time*. *Activities* have a compositional nature, i.e., fine-grained activities make up more general ones. This approach reflects the pragmatic philosophy that the meaning of a place depends mainly upon the activities that occur there, especially the patterns of lower-level activities. The idea applies at both the individual and collaborative level. Such hierarchical context enables generalization or specialization of conditions that apply for a policy rule.

A. Presence context using Nearby

The Nearby APIs were created by Google for creating interaction patterns with objects that are in a devices vicinity [24]. For presence context detection, we have defined a relationship in the Platys ontology as *sitsIn*. This relationship allows us to define that a person has an office room assigned to them in an organization. The subject of this relationship can be a “Supervisor”, “Subordinate” or “Colleague”. The object of the relationship can be a “Location_Room”. In order to obtain this information, we use nearby messaging API from Google that allowed us to deploy a Physical Web of low energy Bluetooth beacons. An example of the utility of such a web or physical infrastructure is the Carlton project [23] which was used to achieve privacy of the organization while responding to natural language queries made about entities in the organization. Using this technique we are able to generate the notion of “User is in front of her Boss’s cabin”, which then allows us to execute policies that contain antecedents that represent presence constraints.

Definition 5: A policy rule VIOLATION, is recorded when a rule defines an access restrictions for an application and a behavior is observed that tries to defy such a restriction.

For the violation detection process, MithrilAC detects application launches and actions performed by the app. Since policy rules in our framework are written as a function of context, the middleware is able to determine if an action performed by an application in a semantic user context is allowed by a currently active policy. If the action is not permitted, it marks the action as a potential policy violation and presents to the user information about the event. If the user feedback is to block the action in question in the future, then we have captured a “True Violation” (TV for short). On the other hand, if the user wants to allow the action or wants to change the currently active policy, a “False Violation”(FV for short) has been captured. When we capture a false violation, the user is allowed to add/delete/generalize or specialize the conditions for a policy rule. The hierarchical context ontology enables the generalization or specialization of rule conditions. Assuming true violations as true positives and false violations as false positives, we can then compute the *precision* of the current policy as follows:

$$VM = \frac{TV}{FV + TV}$$

.

We refer to policy precision as the “Violation Metric” (or VM metric), throughout this work. The VM metric computes the precision of the known policy, as in the ratio of true positives and sum of true and false positives. Here, we are defining true violations as true positives, which signifies that the default policy *P* and the user’s preferred policy *P’* were the same and *NO* modifications to the original policy will be required. On the other hand, false violations or false positives are situations when the default policy *P* and the user’s preferred policy *P’* differ and we need to capture change in current policy. A high value of the VM metric signifies we are closer to a user’s “personalized” policy.

As discussed in section Section II, user driven policy capture processes have shown some promise. However, they have also faced challenges of user indecision, perhaps originating from a lack of understanding about an application’s behavior [29]. Deciding that the system has indeed captured the access control preferences of an individual user thus becomes a key challenge. We present a violation detection driven process as a way to determine when the policy capture process is complete. Violations are actions performed by apps (due to user action or otherwise) that are prohibited by current known policy. We argue that when an such a violation is recorded in a specific context, our current policy requires a modification in order to correctly represent user’s preferred restrictions. A view of the hierarchical choices presented to the user can be seen in Figure 6.

B. User Feedback Algorithm

The USER FEEDBACK ALGORITHM (see Algorithm 1) helps us capture modifications to the initial policy. It uses the hierarchical contextual options encoded using the Platys ontology to capture the afore-mentioned modified policy *P’*. Through this algorithm, we enable the user to accept current rules or modify them by adding, deleting or changing contextual conditions in which they apply.

C. Challenges for detecting app activity

There are two different ways that can be used to detect mobile activity on an Android device. Method one requires root privileges and reads the system log to detect the launch of an activity by monitoring the ActivityManager class using the following command through a shell:

```
logcat -d ActivityManager:I *:S
```

We used the second method, which analyzes the device’s usage history and statistics obtained through Android’s Usage Stats API. However, getting the information about what activity an application performed is still not accessible through this API. For that reason, we had to use the AppOps API. It is important to note that although using the AppOps API we may control the mobile device, getting access to this API is challenging. Moreover, the AppOps API also does not implement context-dependent access control.

The AppOps API is a hidden API and unavailable in the standard Android SDK, Which means our first challenge was that our code for the MithrilAC middleware would not even

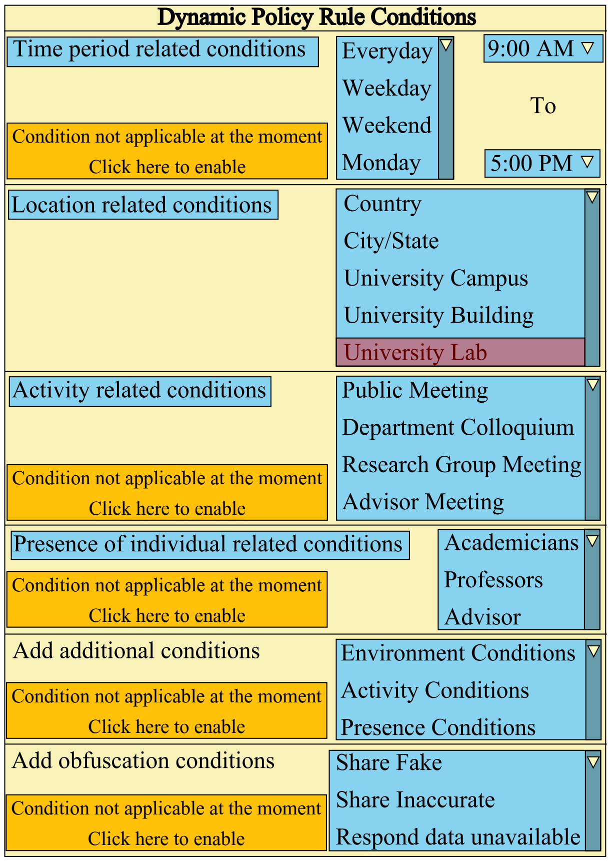


Fig. 6. Ontology-driven hierarchical options for rule modification

compile, if we used the public SDK. To access these hidden APIs one may use Java reflection API but that can slow down the application [30]. There are alternative solutions to using Java reflection, the simplest which is to extract the Android framework JAR file from a real device using the command:

```
adb pull /system/framework/framework.jar
```

The framework.jar is the runtime archive containing all the classes that are used in the android.jar in the SDK. However, the jar contains the runtime optimized version DEX format files. So we have to use dex2jar [31] to convert them to class files by using the command:

```
dev2jar classes.dex
```

Next, we extract everything from our target SDK (in our case API version 25) version's android.jar file from the path `ANDROID_SDK/platforms/android-25/` to a folder and overwrite it with the classes from the framework.jar file folder. Finally we compress the modified classes into a jar file to obtain our "hacked" Android SDK android.jar file with access to the hidden AppOps API.

The second challenge was to detect application actions, as in what resources the application used. Android's documentation does not explain this, but we were able to discover that we can use the AppOps API to detect application actions. Hidden APIs in Java are often accessed using reflection. However, reflection can be tedious and slow [30]. As an alternate

Algorithm 1 "User Feedback Algorithm" - Capture user-specific access control policies

```
1: $appsOnMobileDevice$=get apps on mobile device
2: for each $appsOnMobileDevice$ do
3:     Observe application launches.
4:     Capture resource requested by app.
5:     Collect snapshot of context.
6:     Find out policy rules that deny resource access to application
       in current context.
7:     Store potential violations in for deferred user feedback.
8: end for
9: for each $recordedViolations$ do
10:    if User denoted as false violation then
11:        Ask user to modify rule.
12:        if User wants to add a condition to the rule then
13:            Let user choose one of the conditions to add.
14:        else if User wants to delete a condition from the rule then
15:            Let user choose one of the conditions to delete.
16:        else
17:            if User wants to generalize a condition then
18:                Provide user with a more generic condition as
                   defined by the Platys ontology.
19:            else
20:                Provide user with a more specific condition as
                   defined by the Platys ontology.
21:            end if
22:        end if
23:    else
24:        User denoted as a true violation.
25:    end if
26: end for
```

solution, we were able to use the above-mentioned "hacked" SDK and root privileges to gain access to application actions.

V. Experimental Results

Evaluation of the MithilAC system and the feasibility of using it for capturing personalized access control policies was carried out through a user study conducted over six weeks. During this period rooted mobile devices running a custom LineageOS Android ROM were provided to users. The study was conducted with 24 subjects and each used the system for a period lasting at least seven days and at most 30 days. To detect an app's launch and resource consumed by the application requires special privileges and access to certain Android APIs that are unavailable on official versions of Android. This is why we had to use LineageOS, which can be used to enable access to APIs that are inaccessible on an official Android build from Google.

A. *User study: round 1 results*

In the user study, we ran our experiments on mobile devices with LineageOS 14.1.1, which is equivalent to Android 7.1.1. The study was done in two phases. One of the challenges we faced while carrying out the user study was the wide gap between the number of violations that required user input and the actual number of inputs we received. This issue occurred due to the fact that we used a default deny initial policy. Understandably, this caused a large number of violations. Our argument behind using a default deny policy was that if the system did not have any policy rules at all, a safe, but not necessarily good policy, would be to block events by default. We also wanted to test the feasibility of using the violation detection based approach to capture rules from scratch. Figure 8 shows variation of the VM metric for ten users over a period of several days. We conclude from the first round of the user study that a quasi-safe "deny by default" policy is actually not a good policy from the perspective of usable privacy and security.

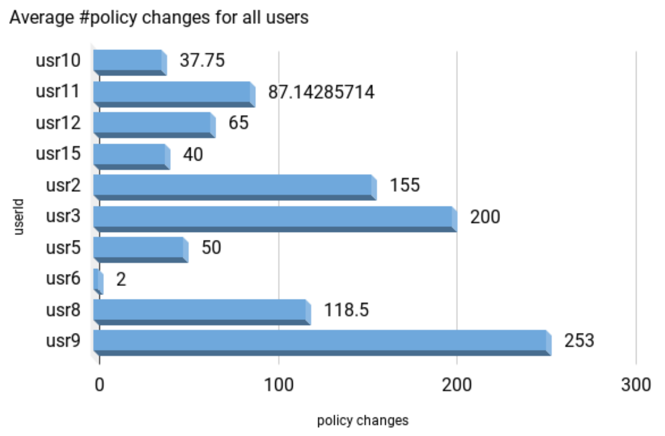


Fig. 7. Average number of policy changes made per user in round 1 of the user study

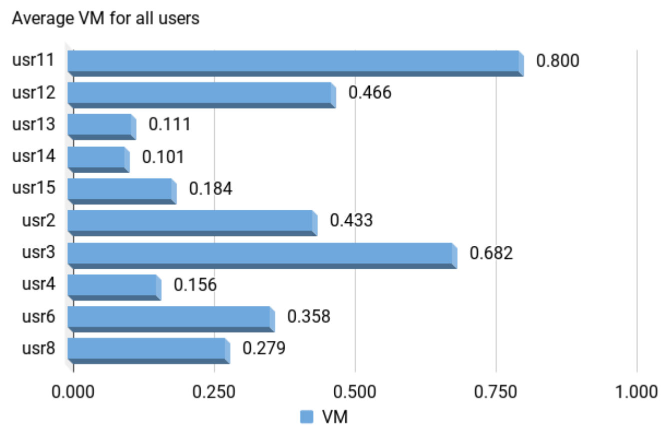


Fig. 8. Average "Violation Metric" per user across multiple iterations in round 1 of the user study

B. User study: round 2 results

Following the conclusions of round one of the user study, we modified our methodology to include a crowdsourced policy. Our crowdsourced policy was originally collected by the Xprivacy open-source app [32]. The data can be found at the link `https://crowd.xprivacy.eu/`. We downloaded approximately 21 million rules for 17k applications and then used category-wise majority voting process to create an initial default policy for the second round of our study. The crowd-sourced policy is obtained by the MithrilAC middleware by querying the back-end server. The results of the second round of the user study are shown in Figure 9.

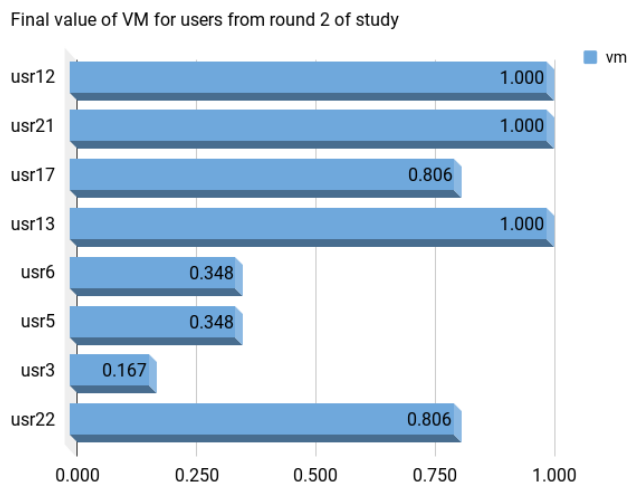


Fig. 9. Average "Violation Metric" per user across multiple iterations in round 2 of the user study

A comparison of the number of violations and the number of feedback inputs from the user can be seen in Figure 7. This graph shows that when using a default deny policy, the number of violations captured were high and the number of times users did not respond to such violations in the feedback loop was also high. However, as soon as we started using the crowdsourced policy as our initial policy generation technique, we saw a drop in number of violations and the number of times users did not respond to our queries were negligible.

C. Reduction in user interaction required

We did the user study in two rounds with a total of 24 users, all of whom were graduate students from UMBC's Computer Science and Electrical Engineering department. The maximum number of policies that were created in round 1 for a user was 3200 and that number came down to 800 in round 2. The average number of applications per user in the study was 48. Table I shows the number of users, violations, true and false violations for the two rounds of user study. From the table we can observe that there is a drop in the number of violations occurring in round 2 of the user study. At the same time we also observe a favorable value for the "Violation Metric" in round 2. We discuss the significance of these results in Section V-D.

Round	#Users	#True violations	#False violations
1	14	228	550
2	10	300	47

TABLE I
USER STUDY VIOLATION STATISTICS

In round 1 we used a default deny initial policy. This led to a lot of violations and user fatigue. As can be seen in Figure 10, "no-response" count in round 1 was relatively high. As a result, we used a curated initial policy based on data from the the Xprivacy crowdsourced dataset. We use normalized frequency counts for our comparison because the two different phases had varying number of users. The first round ended on July 13 on this chart. The second round of the user study thus shows lower number of violations as well as lower number of "no-response" situation from users. The second round of the user study also show a higher number of users with high value of "Violation Metric". This shows that starting with a curated initial policy leads to fewer situations where a user disagrees with the policy defined. Whereas if a default deny policy is used, even legitimate usage of applications get blocked, leading to a larger number of users disagreeing with the policy and hence results in a lower value of "Violation Metric".

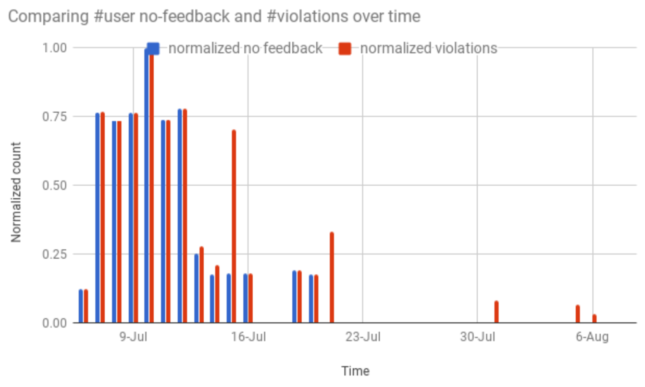


Fig. 10. Comparing user no-response and violations over time

D. *Statistical significance of user study results*

We have presented our results for the user study and application analytics. We next discuss the significance of our results under varying conditions. We performed the user study under two different treatments. Under the first, we used a default deny policy and under the second we used a curated policy generated using crowd-sourced data. Since these two treatments were performed on different sets of users, a paired sample test cannot be performed. We performed an unpaired T-test with the null hypothesis that the two different treatments had the same mean violations and non-responsive situation for users. We were able to reject the null hypothesis through our test, since the computed p value was 0.0033.

VI. User study discussion

A. *Default deny policy*

We performed the user study in two rounds, the first of which did not use a curated policy and assumed a default deny action. As one might guess, this caused a problem for the users who were participating in our study. Upon installation, the MithrilAC middleware started detecting many "violations." These violations were not necessarily true violations as per our definition from Section III. It quickly became evident that a user would find it difficult to provide feedback to all the scenarios that our middleware was presenting to them.

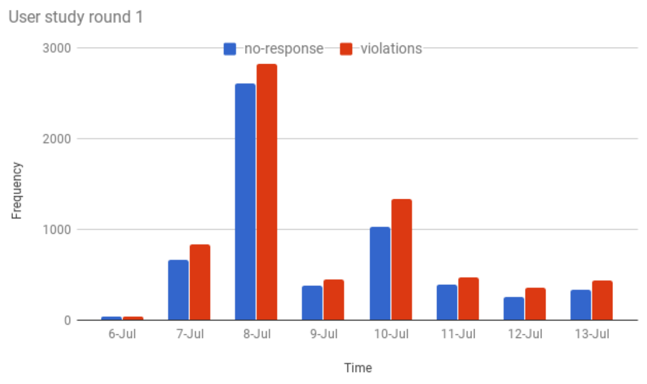


Fig. 11. Comparing #violations and #no-response in user study round 1

B. *Crowd-sourced policy*

Following the issues of round 1 of the user study, we decided to use our crowd-sourced data to generate an initial default policy. We restricted the initial policy generation for just a few contextual situations. Essentially we wanted to show through the second round of our user study that it is feasible to use the violation detection methodology to determine policy deviations for specific users.

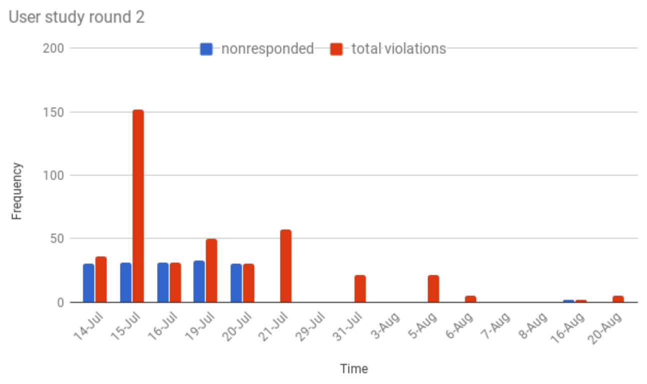


Fig. 12. Comparing #violations and #no-response in user study round 2

The crowdsourced policy lead to a smaller number of violations. A comparison of two rounds of the user study shows that the number of violations went down in the second round when a curated policy was used. In contrast, the first round of user study caused many more violations to occur. At the end of the user study, participants were allowed to respond to a series of questions. A Likert-type scale was used for the questionnaire in order to understand the following questions/issues:

- Do mobile applications take too many permissions and the purpose unclear for a requested permission?
- Are mobile operating system security and privacy settings difficult to find?
- Are application knowledge and ratings useful in making my sharing decision?
- Are user confident in their ability to manage mobile privacy and security settings?
- Does knowledge that an application used a resource in a specific context makes it easy to allow or deny such access?
- Are user privacy and security needs context dependent, for example, allowing social media access at home, but not at work?
- Did the final captured policy better represent? Was using MITHRIL worth the effort given the benefits?

Users were also allowed to provide feedback using the form. Two example responses are shown below:

User 1: "As a naive android user, I did not understand all the meanings of the names of the permissions. Its hard to assess the impact of denying the permission to an application too. The option for whats allowed and what is blocked in custom controls is unclear. Semantics of the privacy setting can be more clear. There is a difficulty understanding the time of old notification. Permissions requested in prior day can no longer be configured since the exact time unclear. What does allowed ignore and running mean? All in all its a good starting effort and the setting requires more explanation and further ease to configure."

User 2: "MITHRIL is great application and has allowed me to understand my privacy requirements better. I was surprised by the permissions, by the number of permissions that were asked by apps. For example, I am not sure why Wikipedia needs my contacts, call log and calendar details. I was happy that MITHRIL allows context modeling. It is more useful for some applications than others. For example, I would not mind if Waze asks for contact information and call log access when I am driving. But, I would not want it to access that information otherwise. Having context is helpful. I also like that number of feedbacks asked by MITHRIL reduced over time. Following are few things that will make application more helpful: 1. It should have more contexts like 'Driving'. 2. Rather than having separate context window, I would prefer to add context whenever needed in customize button. It would be nice to have everything in same window, rather than having separate window for customization."

Responses to the feedback questions show that users do feel applications are intrusive and controlling while they are unsure why they have requested a permission while finding permission settings seems difficult to them. 44% of the users were neutral towards application ratings to decide access decisions while most of them were confident in making such decisions. Context plays a role in their access decisions and knowledge of application activity helps in making allow/deny decisions for users. Finally, users felt the final policy was representative of their needs and using the system was worth the effort given the benefits.

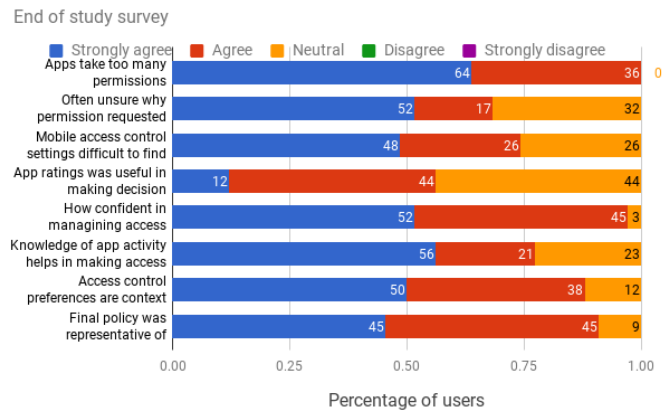


Fig. 13. User feedback to questionnaire

VII. CONCLUSIONS

We have presented the MithrilAC middleware system, a component of MITHRIL, an end-to-end context-dependent access control framework that monitors mobile application activities on a user's mobile device in various contextual situations and captures their access control preferences in that context. We have presented the implementation of MithrilAC and a user study that proves the feasibility of using precision of captured policy to determine completion of user policy capture process. The main contributions of our work is the design and development of the MithrilAC system that achieves the following goals:

- Create mobile-middleware that uses human-in-the-loop collaborative editing of access control policies.
- Develop middleware that displayed run time app activity to user in order to assist them in edit their policy preferences.
- Evaluate the feasibility of using "Violation Metric" as a way to determine completion of policy capture process through a User Study.
- Show that a curated initial default policy reduced user interaction required.

Through our user study experiments we can conclude that using a default deny policy creates too many policy violations and causes user fatigue leading to low number of responses. However, if a curated initial default policy is used, we see a reduction in user burden for policy customization. This happens because the user does not have to make too many changes for such a policy if it is collected from an appropriate source. In our previous work [5], we show how to create such a policy. A discussion on the creation of such a policy is beyond the scope of the current paper.

One of the most difficult task, that we had to perform was to get a working system that captures actual events on a mobile device and feedback from real users about those events. Given our collected data, an obvious future work is to use pattern recognition and machine learning algorithms to predict a user's preference choices. Improvements to the behavior classification process is an important goal that we hope to pursue in the future. Since static features improved classification accuracies combining them with dynamic and network features could lead to better application behavior classifications. Some of the suggestions made by users in our study could further improve the usability of the system. We hope to incorporate these in the future. Another potential future work includes building a policy that blocks applications that are determined to be "too unsafe" through the risk computation. Finally, we would like to perform study of things that were allowed in policy and blocked by the operating system.

VIII. Acknowledgment

Support for this work was provided by NSF grants 0910838 and 1228198.

References

[1] I. Security, "Mcafee lbs: Threats report," November 2014. [Online]. Available: http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q3-2014.pdf

[2] E. Root, A. Polkovnichenko, and B. Melnykov, "Expensivewall: A dangerous packed malware on google play that will hit your wallet," September 2017. [Online]. Available: https://blog.checkpoint.com/2017/09/14/expensivewall-dangerous-packed-malware-google-play-will-hit-wallet/

[3] Google, "The google android security teams classifications for potentially harmful applications," April 2016. [Online]. Available: https://goo.gl/Ez1ojF

[4] P. Kumaraguru and L. F. Cranor, "Privacy indexes: a survey of westin's studies," *School of Computer Science, Carnegie Mellon University, Pittsburgh*, 2005.

[5] P. K. Das, "Context-dependent privacy and security management on mobile devices," 2017.

[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*, 2004.

[7] P. Jagtap, A. Joshi, T. Finin, and L. Zavala, "Privacy preservation in context aware geosocial networking applications," *organization*, 2011.

[8] M. Dean, G. Schreiber, S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein, "Owl web ontology language reference," *W3C Recommendation February*, vol. 10, 2004.

[9] S. Aich, S. Mondal, S. Sural, and A. K. Majumdar, "Role Based Access Control with Spatiotemporal Context for Mobile Applications," *Transactions on Computational Science*, vol. 4, pp. 177–199, 2009.

[10] V. C. Hu, D. Ferraiolo, R. Kuhn, A. R. Friedman, A. J. Lang, M. M. Cogdell, A. Schnitzer, K. Sandlin, R. Miller, K. Scarfone *et al.*, "Guide to attribute based access control (abac) definition and considerations (draft)," *NIST Special Publication*, vol. 800, no. 162, 2013.

[11] 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*, 2012, pp. 82–85.

[12] L. Kagal, T. Finin, and A. Joshi, "A policy language for a pervasive computing environment," in *IEEE 4th International Workshop on Policies for Distributed Systems and Networks*. IEEE, 2003, pp. 63–74.

[13] M. Benisch, P. G. Kelley, N. Sadeh, and L. F. Cranor, "Capturing location-privacy preferences: Quantifying accuracy and user-burden tradeoffs," *Personal and Ubiquitous Computing*, vol. 15, no. 7, pp. 679–694, 2011.

[14] 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.*, vol. 13, no. 6, pp. 401–412, Aug. 2009.

[15] 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*. Menlo Park, CA: USENIX Association, Jul. 2014, pp. 199–212.

[16] 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*, ser. WWW '14. New York, NY, USA: ACM, 2014, pp. 201–212.

[17] B. Liu, M. S. Andersen, F. Schaub, H. Almuhimedi, S. Zhang, N. Sadeh, A. Acquisti, Y. Agarwal, B. Liu, M. S. Andersen, F. Schaub, H. Almuhimedi, S. Zhang, N. Sadeh, A. Acquisti, and Y. Agarwal, "Follow My Recommendations : A Personalized Privacy Assistant for Mobile App Permissions," in *Symposium on Usable Privacy and Security*, 2016.

[18] L. Zavala, R. Dharurkar, P. Jagtap, T. Finin, and A. Joshi, "Mobile, collaborative, context-aware systems," in *Proc. AAAI Workshop on Activity Context Representation: Techniques and Languages, AAAI. AAAI Press*, 2011.

[19] K. Lee, J. Lee, and M.-P. Kwan, "Location-based service using ontology-based semantic queries: A study with a focus on indoor activities in a university context," *Computers, Environment and Urban Systems*, vol. 62, pp. 41 – 52, 2017.

[20] H. Chen, T. Finin, and A. Joshi, "The SOUPA Ontology for Pervasive Computing," *Computing Systems*, pp. 233–258, 2005.

[21] D. Ferreira, V. Kostakos, and A. K. Dey, "Aware: mobile context instrumentation framework," *Frontiers in ICT*, vol. 2, p. 6, 2015.

[22] S. Saguna, A. Zaslavsky, and D. Chakraborty, "Complex activity recognition using context-driven activity theory and activity signatures," *ACM Trans. Comput.-Hum. Interact.*, vol. 20, no. 6, pp. 32:1–32:34, Dec. 2013.

[23] P. K. Das, A. Kashyap, G. Singh, C. Matuszek, T. Finin, and A. Joshi, "Semantic Knowledge and Privacy in the Physical Web," *4th Workshop on Society, Privacy and the Semantic Web - Policy and Technology, co-located with the 15th Int. Semantic Web Conf.*, 2016.

[24] Google, "Nearby messages api," May 2017. [Online]. Available: https://developers.google.com/nearby/messages/overview

[25] B. Parducci, H. Lockhart, and E. Rissanen, "Extensible access control markup language (XACML) version 3.0," *OASIS Standard*, pp. 1–154, 2013.

[26] J. Fürnkranz, D. Gamberger, and N. Lavrač, *Foundations of rule learning*. Springer, 2012.

[27] 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.

[28] N. K. Sharma and A. Joshi, "Representing attribute based access control policies in owl," in *2016 IEEE Tenth International Conference on Semantic Computing (ICSC)*. IEEE, 2016, pp. 333–336.

[29] J. Lin, J. I. Hong, B. Liu, N. Sadeh, and J. I. Hong, "Modeling Users ' Mobile App Privacy Preferences : Restoring Usability in a Sea of Permission Settings," *Proceedings of the tenth Symposium on Usable Privacy and Security*, vol. 1, pp. 1–14, 2014.

[30] S. Chiba, "Load-time structural reflection in java," in *European Conference on Object-Oriented Programming*. Springer, 2000, pp. 313–336.

[31] B. Pan, "dex2jar," 2015. [Online]. Available: https://github.com/-pxb1988/dex2jar

[32] M. Bokhorst(M66B), "Xprivacy," June 2013. [Online]. Available: https://github.com/M66B/XPrivacy