User Controllable Security and Privacy for Mobile Mashups

Shruthi Adappa[1], Vikas Agarwal[2], Sunil Goyal[2], Ponnurangam Kumaraguru[3], Sumit Mittal[2]
[1]*Georgia Institute of Technology, Atlanta, USA*
shruthiadappa@gatech.edu
[2]*IBM India Research Lab, New Delhi, India*
{avikas, gsunil, sumittal}@in.ibm.com
[3]*Indraprastha Institute of Information Technology, New Delhi, India*
pk@iiitd.ac.in

ABSTRACT

A new paradigm in the domain of mobile applications is 'mobile mashups', where Web content rendered on a mobile browser is amalgamated with data and features available on the device, such as user location, calendar information and camera. Although a number of frameworks exist that enable creation and execution of mobile mashups, they fail to address a very important issue of handling security and privacy considerations of a mobile user. In this paper, we characterize the nature of access control required for utilizing device features in a mashup setting; design a security and privacy middleware based on the well known XACML policy language; and describe how the middleware enables a user to easily control usage of device features. Implementation-wise, we realize our middleware on Android platform (but easily generalizable to other platforms), integrate it with an existing mashup framework, and demonstrate its utility through an e-commerce mobile mashup.

Keywords

Mobile Mashups, Device Features, Security and Privacy

1. INTRODUCTION

Rapid enhancements in computing power, memory, display, etc. have propelled mobile phones as a platform to deploy and execute a variety of applications. On most modern smart mobile platforms, native device features (e.g. location, calendar information, camera) are available as easy to use *APIs* (Application Programming Interface) within mobile applications. Developers embed these APIs within their applications to enrich and enhance user experience. Next generation 'mobile mashups' [9] pertain to the class of applications that let these APIs be invoked from within Web content rendered on the mobile browser through JavaScript based interfaces. For example, one can embed the facility to invoke device camera, click pictures and tag them with the current GPS location before uploading and sharing, all within the context of a Web-based social networking application. A number of mashup frameworks exist [1] that expose mashable interfaces to developers for various device features.[1] [2] Although exact implementations differ, all these frameworks essentially work by providing a bridge between JavaScript invocations in a browser-rendered mashup and the underlying native interfaces in C, Java, etc.

As mobile phones evolve into smart-phones, security and privacy around devices' novel capabilities are critically important, both, in an individual setting, and in an enterprise one. On the one hand, user's data, such as location, as well as confidential information, such as contacts and calendar entries, are sensitive and should not be shared with malicious agents. On the other hand, invocation of certain phone features, for instance, SMS, MMS and Call, incur cost and utmost care needs to be taken to prevent unauthorized access to the same. Similarly, starting of the camera without the user's desire can result in undue privacy invasion and drained battery. In the case of mobile mashups, the problem gets compounded further as content rendered belongs to a third party site whose services might have been compromised. Security and privacy aspects are not adequately addressed in current mashup frameworks. For example, PhoneGap delegates these issues to the device platform, while OMTP Bondi gives flexibility to a mashup developer for exercising control during creation, but not to the mashup user at the time of execution.

The requirements of an access control mechanism involving device features in a mobile mashup are vastly different from what is possible through provisions in the browser or platform OS. For example, a user might want to restrict the set of features that mashups from a particular domain can access. Further, control for each feature itself might be desired at a much finer level – such as allowing location information to be revealed to an enterprise mashup only during office hours; limiting the number of Calls made or SMSes sent in a day by a particular mashup; putting a cap on frequency of access with respect to different features, etc. In essence, invocation of features should be decided both by the *mashup context* (e.g. which site is hosting it) as well as the *user context* (e.g. current time of day). This is not possible through the mobile browser that at best can be configured to block or allow certain mashup sites. Some mobile platforms, such as BlackBerry and iPhone allow users to configure access (allow, deny, prompt) to device features, but it is too

[1]http://phonegap.com/
[2]http://bondi.omtp.org/

coarse-grained and mostly consists of prompting the user every time a device feature is invoked. While annoying the user, it is also exposed to vulnerabilities [7].

Various policy languages exist (XACML [10], DPAL [3], P3P [4], etc.) that help in managing privacy and security requirements – expressing, analyzing, combining, and enforcing. In this paper, we take a comprehensive view of user controlled security and privacy for mobile mashups, analyze suitability of policy languages to our context, and propose a middleware for empowering mashup users to control usage of device features. In particular, key contributions of this paper are:

- We define the access control requirements for mobile mashups and identify a suitable policy language to express access control policy.
- We design a security middleware based on XACML policy language for controlling access to native device features and data.
- We realize our middleware on the Android platform and integrate it with an existing mobile mashup framework.

Even though we concentrate on Android with respect to implementation, the middleware can be adapted to other platforms like iPhone, BlackBerry, Nokia S60, as well as integrated with other mashup frameworks (PhoneGap, OMTP Bondi). To help users deal with policies, we also develop a policy configuration utility for adding and editing policies. The utility of our middleware is demonstrated through an e-commerce mashup.

2. DESIGN OF SECURITY AND PRIVACY MIDDLEWARE

In this section, we begin by characterizing access control in the context of mobile mashups. Thereafter, we outline a security and privacy middleware that realizes our requirements through a well known policy language, and also describe a generic model for integration of this middleware with existing mashup frameworks.

2.1 Access Control Requirements

As described earlier, the requirements of an access control mechanism in a mobile mashup is different from that in the browser or platform OS. To build a middleware that enables appropriate access control for device features, we first determined the set of attributes that are needed for decision making. The broad criteria we had was that the attributes should encompass both the 'mashup context' and the 'user context', as mentioned in Section 1. Mashup context refers to the set of attributes pertaining to a mashup, consisting of the mashup URL, device features being accessed, and frequency of access. On the other hand, user context comprises of attributes specific to a user, such as time when the mashup is being accessed, current location of user, calendar information, contact list, etc. In essence, we wish to cater to settings where a user may specify that a call can be made only to numbers in his contact list or that Camera cannot be invoked while he is in office. Based on both the mashup context and the user context, there can be four possible outcomes while making decision on request for a device feature – allow, deny, abstracting the information (e.g., providing approximate location instead of the exact latitude and longitude), or prompt the user for further information [6].

We observe that various security and privacy requirements for a mashup can be represented as policies, and enforced through a policy execution engine. Instead of designing a new policy language, we explore suitability of existing languages for our purpose. This is described next.

2.2 Choice of Policy Language

Different policy languages are available to represent human-readable policies in more precise and computer compatible formats. Some languages are designed to help *organizations* express their security and privacy policies whereas others are more amenable for *users* to define their preferences. Each language has its own syntax and mechanisms for implementation. The World Wide Web Consortium (W3C) developed the Platform for Privacy Preferences (P3P) to express website privacy policies in machine-readable format [4]. eXtensible Access Control Markup Language (XACML) [10] was designed by a consortium of organizations for expressing both privacy and security policies in a machine-readable format. There were other initiatives such as DPAL [3], and XPref [2]. Kumaraguru et al. present a comprehensive understanding and comparison of these and other privacy policy languages [8].

After a careful analysis of the existing policy languages, we chose XACML for our purpose. The reasons for this are as follows – (i) XACML supports basic access controls (allow, deny, inapplicable, and indeterminate), (ii) it is an industry standard access control policy language and supports distributed policies, (iii) it is domain independent, (iv) it can be customized to be light-weight depending on the features that one needs to implement, and (v) there are several publicly available implementations of XACML, in various programming languages, including at least one open source implementation. These features best fit our requirements.

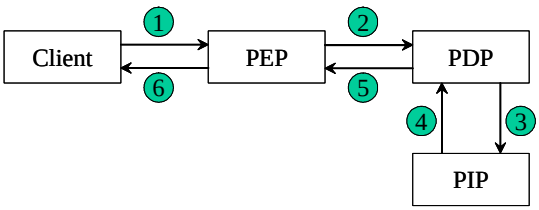


Figure 1: Main XACML entities

XACML describes a policy language and an access control decision request/response language, both encoded in XML. Figure 1 depicts a part of the XACML architecture consisting of four entities - Client, Policy Enforcement Point (PEP), Policy Decision Point (PDP) and Policy Information Point (PIP). Client application desires to access a service protected by PEP. On receiving a request from Client (1), PEP creates an XACML request that contains Client identifier (*Subject*), *Resource* identifier and *Action* requested (e.g. read or write). This request is sent to PDP (2) for evaluation using a predefined Policy Set. PDP requests PIP (3) for attributes associated with the Subject / Resource / Action / Environment. PDP then evaluates the request on receiving attributes from PIP (4) and sends its decision or response to PEP (5). Access to the client is given accordingly (6).

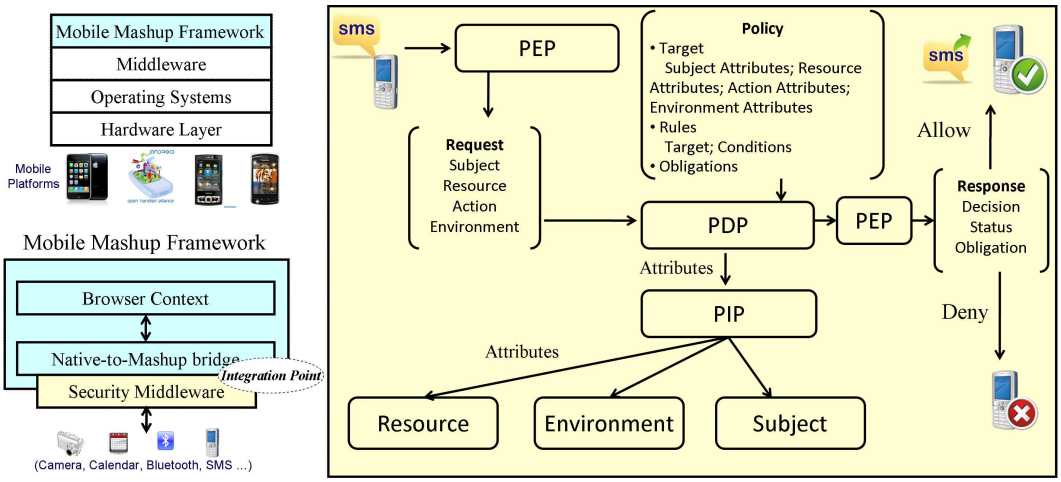


Figure 2: Security and Privacy Middleware. *Top left*: Presents the modular architecture for a mobile platform. *Bottom left*: shows details of the mobile mashup framework (adapted from Agarwal et al. [1]), and integration of security middleware. *Right*: shows how we used the XACML building blocks in our security middleware.

The Response consists of a Result object, which represents the result of an evaluation. This Result object contains a Decision (Permit, Deny, NotApplicable, or Indeterminate), Status information (for instance, why evaluation failed), and optionally one or more Obligations (things that the PEP is obligated to do before granting or denying access).

2.3 Realizing Security and Privacy Middleware

We first express our security and privacy requirements discussed in Section 2.1 through policies using constructs provided by the XACML policy language. Thereafter, these policies are enforced using an XACML engine. To understand this process, consider a scenario where a user in the context of a mashup application wants to control the usage of SMS API (e.g., based on time of day, intended recipient). When the application tries to send an SMS, the process implemented in our middleware would be as follows (refer Figure 2):

- the mashup framework makes a request to the native platform for accessing the SMS API.
- the request is passed to PEP, where it receives URL of the application, the called API method, `sendTextMessage`, and the parameters of the method. PEP then packages the request into an appropriate format to build a Request Context object.
- the constructed object is given to PDP; here the request is evaluated with the list of applicable policies obtained from the policy module.
- support for external attributes not found in the policy is implemented using PIP. In our case, this means that PIP dynamically retrieves the user context (e.g. current time, contact list) and makes it available to PDP for decision making.
- access control decision of permit, deny, inapplicable or indeterminate is returned back from PDP to PEP.
- PEP then based on the decision returned by PDP decides whether to allow access (call the appropriate API method) or deny access. In our middleware, we assume that a decision of Deny, Indeterminate, or Not Applicable all result in a denial of service.
- the call finally returns to the framework either allowing access to the API or throws an appropriate exception.

Although the above process demonstrates controlling usage of SMS, it applies generally to all device features. Moreover, each feature is controllable via separately specified access criteria. Some Mobile OS, such as iPhone and BlackBerry, allow users to restrict usage of certain device features through applications. Our middleware works on top of it, therefore it does not override restrictions defined by the user through underlying OS. At the same time, it allows for defining fine-grained policy for device feature access.

2.4 Integration with Mashup Frameworks

Mobile mashup frameworks use a 'Native to Mashup bridge' to provide access to local device features, such as Camera, GPS Location and SMS, from within the web content rendered on the browser context. As described in [1], this bridge provides a bidirectional communication between the browser context and the native platform, and exposes device features using JavaScript APIs. In order to ease integration of our security middleware with a given mashup framework on a given mobile platform (Android, iPhone, Blackberry, Nokia S60), a generic mechanism can be used that utilizes the point where a mashup framework talks to the native platform, i.e. the bridge, as shown in Figure 2.

We propose two options for integrating our security and privacy middleware with a bridge – (i) it can reside within the bridge module, or (ii) it can reside between the bridge and the native platform. The first option is possible if one has access to the source code of a mashup framework. On the other hand, second option is useful if source code is not available or it is desired to keep the bridge code separate from the one that enables access control to device features. In this case, security and privacy can be classically considered as 'aspects' that are woven with the core mashup framework by way of intercepting calls for native features.

It should be noted that the middleware needs to be integrated only with the bridge, and not directly with the browser, as it potentially requires current user context while evaluating the request (e.g. getting the current location or contact list); this would require capabilities of the bridge.

3. IMPLEMENTATION

In this section, we first describe an Android implementation of our security middleware and its integration with a mashup framework. Then we describe a policy configuration utility that we have built to help users configure various policies for mashup applications.

3.1 Android Implementation

We first took desktop implementation of Sun's XACML, [3] an open source Java implementation of the OASIS standard, and ported it to run on Android (version 2.2) platform. We modified it so that the policy files are read from a local data folder of the middleware. This ensures that no other application can access these policy files and modify them.

When the security middleware receives a request from the mashup framework for a device feature, it constructs a request object from the parameters obtained from the JavaScript call. These parameters are passed to the `RequestBuilder` object which calls appropriate methods to setup the Subject (URL of the mashup), Resource (API to be called), Action (frequency of calls made per hour), and Environment (time of call). This results in the creation of a `RequestCtx` object which is then passed as a parameter to the `PDP.evaluate()` function that performs the actual policy evaluation and returns a `ResponseCtx` object containing the relevant decision of allowing or denying the request.

In our implementation, URL of a mashup is used to identify the policy for that mashup. In particular, we keep a mapping from the mashup URL to the policy filename in a configuration file. Since there might be multiple mashup applications available from a given organization (domain), a user would possibly want to have a single policy file for all these mashups. To support this, we use the longest prefix matching of URL to find the appropriate policy file. Note that if the user has configured two set of policies – one for a mashup (using the full mashup URL) and another for domain (using only the domain URL like http://www.ibm.com/), conflict is resolved in favor of the former. This gives flexibility to the user in terms of configuring a single domain-level policy that is applicable to multiple mashups from that domain, and if required, configuring a separate policy for a specific mashup. Enterprises can also use this feature to configure an enterprise-wide policy.

3.2 Integration with Mashup Framework

In our earlier work, we have implemented a mobile mashup framework for various platforms, including Android [1]. On Android, this framework uses the `addJavaScriptInterface()` API of the `WebView` class to create the native-to-mashup bridge discussed earlier. The `accessNativeFeature()` API of this framework is a wrapper for this bridge and is used to call various device features (Location, Contacts, Camera, etc.) by passing the corresponding API name and the associated request parameters.

[3]http://sunxacml.sourceforge.net/

For integrating our security middleware with this framework, we first intercept calls to the native functionalities in the `accessNativeFeature()` method of the framework object. Next, we call the `evaluateRequest()` method of the security middleware and pass the request parameters. As described above, this method checks whether access to this functionality is allowed or denied. If access is allowed then the rest of the `accessNativeFeature()` method is executed, otherwise an 'access denied' message is returned to the user.

3.3 Policy Configuration Utility

We built a GUI-based policy editor to allow users to easily specify their access control preferences for mashup applications. In the current version, for each mashup, a user can specify the frequency of access (per hour) and time range during which access to an API is allowed. The editor can be used in two modes: *Add* and *Edit*. In the Add mode, users can create a new policy for an application. On entering the URL of the application and clicking on the 'Save' button for an API, a new policy is created if a policy file does not already exist for that URL, else the existing policy file is opened and a new rule is added for that API. When a new policy file is created, the file name along with the URL mapping is stored in a configuration file. In the Edit mode, on choosing a URL and an API, users can view the current time range and frequency values, and edit them if needed. Figure 3 presents a screen shot of the configuration utility. It shows how to configure access to *Location* API from 5:00 PM to 10:00 PM for any mashup application from www.ibm.com domain. Figure 4 shows the corresponding XACML policy fragment.

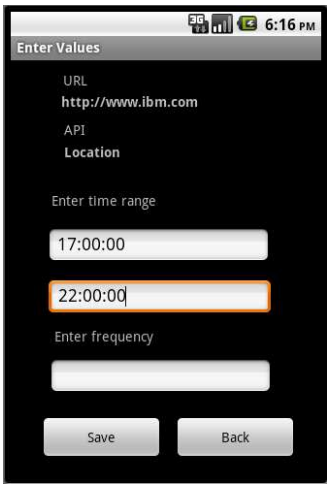


Figure 3: Policy Configuration Utility

4. DISCUSSION

To demonstrate the utility of our middleware, we developed an e-commerce mashup application that lets a user scan the barcode of a product, send it over to a server along with the user's current location, and determine if the product is available at any store in the vicinity of the user. The mashup also displays the location of nearby relevant stores on a map, with information on promotions and offers, and provides facility to contact the stores via SMS or phone call. Note that the eCommerce mashup brings together various

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:1.0:policy"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" PolicyId="Scenario1"
        ...
        <Rule RuleId="Location" Effect="Permit">
        <Target>
        <Subjects>
                <AnySubject/>
        </Subjects>
        <Resources>
                <Resource>
                <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
        equal">
                <AttributeValue
        DataType="http://www.w3.org/2001/XMLSchema#string">Location
                </AttributeValue>
                <ResourceAttributeDesignator
        DataType="http://www.w3.org/2001/XMLSchema#string"
                AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
                </ResourceMatch>
                </Resource>
        </Resources>
        <Actions>
                <AnyAction/>
        </Actions>
        </Target>
        <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
        <Apply FunctionId="http://research.sun.com/projects/xacml/names/function#time-in-
        range">
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-and-only">
                <EnvironmentAttributeDesignator
        DataType="http://www.w3.org/2001/XMLSchema#time"

                AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"/>
        </Apply>
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">17:00:00
        </AttributeValue>
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">22:00:00
        </AttributeValue>
        </Apply>
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-less-than-or-
        equal">
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-one-and-only">
                <ActionAttributeDesignator
        DataType="http://www.w3.org/2001/XMLSchema#integer"
                AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"/>
                </Apply>
                </Apply>
        </Condition>
        </Rule>
        <Rule RuleId="FinalRule" Effect="Deny"/>
</Policy>
```

Figure 4: XACML Policy Fragment

device features (GPS *Location* to determine user's whereabouts, *Camera* to capture barcode, *Call* to start phone call, *SMS* to send text message), enterprise functions (*Catalog*, *Store Directory*, *Competitive Analysis*) and the Web (*Google Maps* for visual display of stores on a map).

Figure 5 presents screen-shots of this eCommerce mashup with policy enforcement. In particular, Figure 5(a) displays a list of stores pertaining to the user's search for a product, Figures 5(b) and (c) depict two possible outcomes (allowed to proceed and blocked, respectively) when the user tries to invoke the 'Call' feature of the device. As mentioned earlier, which of the two outcomes is enforced is determined by policies defined for this mashup.

Currently, our policy editor is designed for only simple policies that are easily configurable by a large number of users. In general, however, setting of policies for a mobile mashup can be a complex affair, given the number of attributes in the mashup and user context. For example, a user might wish to allow phone calls from within a given mashup to only people in his address book. Similarly, an enterprise may restrict the use of Camera while the employees are in office; presence in office to be deduced using employee's current location. As we move towards supporting such complex policies in our editor, we face a trade-off between 'usability' and 'complexity' associated with the editor. Through various user studies, we wish to arrive at a sweet spot where a variety of policies can be configured through the editor without compromising its usability. One feature that we definitely want to support is the provisioning of resource-wide policy, whereby policy defined for a resource is applicable across different mashups. It should also be noted that an enterprise/user always has the option of bypassing the policy editor to manually configure arbitrarily complex policies in the middleware.

The task of setting policies for a number of mashups can become mundane for a user and he may be de-motivated to set a reasonable policy for each new mashup. In order to provide users an easy means, a *learning engine* can be provided. Currently, if the policy for an API is not found, it implies 'deny'. This behavior can be changed to prompt the user to get his choice when a policy for an API is not found. The learning engine can collect and store user responses along with various context data. For a given mashup, based on choices made by the user over a period of time, the learning engine would be able to infer a reasonable policy. The inferred policy can then be presented to the user for verification, editing, and eventual setting. We believe that the learning engine can be a powerful addition to our middleware and will be especially useful when the user accesses a mashup for the first time and no applicable policy exists for the same.

In this paper, we have not addressed threats with respect to a compromised OS or a compromised mashup framework. Though interesting and important, we feel that these issues are orthogonal to our work and need to be investigated in the more general context of mobile system security. A compromised OS poses threat across all user interactions and applications on the mobile, not only to mashups in our framework. Similarly, a compromised framework can override policies set by the user for various mashups, and therefore care should be taken by the user to obtain framework from a trusted entity only.

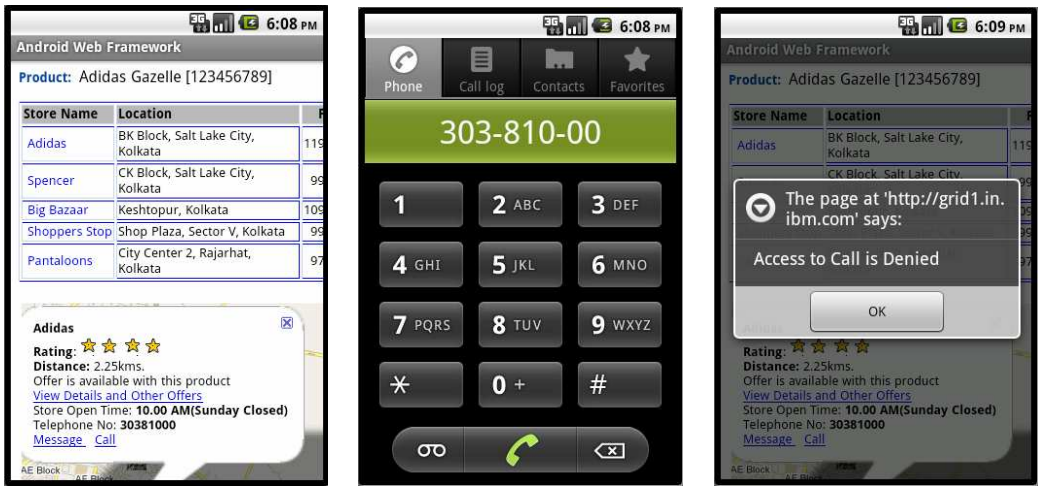
5. RELATED WORK

There are several professional tools in the industry to create Web based mashups like Yahoo Pipes [4], BEA AquaLogic, and IBM Lotus Mashups. [5] However, none of these allow integration of device features for developing mobile mashups. Similar to our mashup framework [1], OMTP Bondi and PhoneGap provide mechanism to add device features in a mobile mashup. Bondi aims to standardize a set of JavaScript APIs for accessing device features. Therefore, semantic and syntactic differences in various APIs are automatically removed across all devices and platforms that implement the Bondi specification.

Bondi also specifies a security framework responsible for controlling access to JavaScript APIs, but does not provide concrete examples of usage scenarios with sample policy files. Over and above, the security model of Bondi is at the JavaScript level, unlike our framework which is implemented in Java (or other native device language) between the JavaScript to Java interface. Thus, our model empowers both the mashup user and mashup developer. The other framework, PhoneGap, defines uniform JavaScript APIs to access device features and implements the native-to-mashup bridge on several platforms. However, PhoneGap neither defines a security model for mashups, nor defines any user

[4]http://pipes.yahoo.com/pipes/

[5]http://www-01.ibm.com/software/lotus/products/mashups/



(a) (b) (c)

Figure 5: Access Control for an eCommerce Mashup Application. (a) shows store list; (b) shows the user interface where 'Call' feature is allowed; (c) shows the user interface where 'Call' feature is blocked.

control over access to device features, essentially delegating the entire process to the device OS.

Hypponen's work [7] was one of the first to identify major privacy and security challenges around use of sensitive user information like location, contacts and calendar entries in mobile applications. Taint-Droid [5] looks inside of the mobile applications to watch how they use privacy sensitive data and monitors their behavior to determine when privacy sensitive information leaves the phone. Currently, various enterprise-grade mobile applications provide in-built access control for specific device features. For instance, PeopleFinder [11] is a location sharing application that gives users ability to create rules with varying complexity for configuring privacy settings around sharing their location. In this paper, we have created an access control middleware that differs on two counts. Firstly, we move beyond location and cover other device features as well. Secondly, we apply policies to a generic mobile mashup setting, and not to a specific application.

6. CONCLUSION AND FUTURE WORK

In this paper, we characterized the nature of access control required for utilizing device features in a mobile mashup setting, designed a security and privacy middleware based on the well known XACML policy language, and described how the middleware enables a user to easily control usage of device features. We developed a Policy Configuration editor that allows users to express, manage and refine security policies. We integrated our middleware with an existing mobile mashup framework on the Android platform, and demonstrated it's utility through an e-commerce mashup.

Going forward, we wish to enhance our middleware along several dimensions. More specifically, we want to devise a learning component and provide support for complex policies in our editor, as discussed in Section 4. Further, in case of multiple applicable policies for a given mashup, we want to explore other conflict resolution schemes apart from the longest-prefix-match one that we currently use. Implementation-wise, we intend to offer support for other popular mobile platforms. Finally, through user studies, we wish to gain valuable insights regarding the usefulness of our middleware.

7. REFERENCES

[1] V. Agarwal, S. Goyal, S. Mittal, S. Mukherjea, J. Ponzo, and F. Shah. Towards Enabling Next Generation Mobile Mashups. In *Proceedings of the 7th International ICST Conference on Mobile and Ubiquitous Systems*, Dec 2010.

[2] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. An XPath-based Preference Language for P3P. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 629–639, New York, NY, USA, 2003.

[3] A. Barth and J. C. Mitchell. Conflict and Combination in Privacy Policy Languages. In *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society*, 2004.

[4] L. Cranor, M. Langheinrich, M. Marchiori, M. Presler-Marshall, and J. Reagle. The Platform for Privacy Preferences 1.0 (P3P1.0) Specification. April 2002. http://www.w3.org/TR/P3P/.

[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 Symposium on Operating Systems Design and Implementation (OSDI)*, October 2010.

[6] S. K. Ghai, P. Nigam, and P. Kumaraguru. Cue : A Framework for Generating Meaningful Feedback in XACML. *3rd ACM Workshop on Assurable & Usable Security Configuration (in conjunction with CCS 2010)*, October 2010.

[7] M. Hypponen. Malware Goes Mobile. *Scientific American*, November 2006.

[8] P. Kumaraguru, L. Cranor, J. Lobo, and S. Calo. A survey of privacy policy languages. *Workshop on Usable IT Security Management (USM '07) at Symposium On Usable Privacy and Security '07*, 2007.

[9] E. M. Maximilien. Mobile Mashups: Thoughts, Directions, and Challenges. *International Conference on Semantic Computing*, pages 597–600, 2008.

[10] T. Moses. eXtensible Access Control Markup Language (XACML) Version 2.0. Technical report, Oasis, 2004. http://xml.coverpages.org/XACMLv20CD-CoreSpec.pdf.

[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. *Journal of Personal and Ubiquitous Computing*, 13(6), August 2009.