Recommendation Models for Open Authorization

Mohamed Shehab, *Member*, *IEEE Computer Society*, and Said Marouf

Abstract—Major online platforms such as Facebook, Google, and Twitter allow third-party applications such as games, and productivity applications access to user online private data. Such accesses must be authorized by users at installation time. The Open Authorization protocol (OAuth) was introduced as a secure and efficient method for authorizing third-party applications without releasing a user's access credentials. However, OAuth implementations don't provide the necessary fine-grained access control, nor any recommendations, i.e., which access control decisions are most appropriate. We propose an extension to the OAuth 2.0 authorization that enables the provisioning of fine-grained authorization recommendations to users when granting permissions to third-party applications. We propose a multicriteria recommendation model that utilizes application-based, user-based, and category-based collaborative filtering mechanisms. Our collaborative filtering mechanisms are based on previous user decisions, and application permission requests to enhance the privacy of the overall site's user population. We implemented our proposed OAuth extension as a browser extension that allows users to easily configure their privacy settings at application installation time, provides recommendations on requested privacy permissions, and collects data regarding user decisions. Our experiments on the collected data indicate that the proposed framework efficiently enhanced the user awareness and privacy related to third-party application authorizations.

Index Terms—OAuth, collaborative filtering, social networks.

1 Introduction

Online platforms have become rich grounds for third-party applications that utilize user online data to provide various services. Third-party applications, especially within social networking platforms have become very popular and pervasive. For example, with over seven million third-party applications on Facebook, its users install applications more than 20 million times a day [8]. Before using applications, users are required to authorize them and grant them access to certain permissions they request, e.g., access to a user's e-mail, location, etc. With the pervasiveness of such applications, protecting the user's online private data becomes a necessity.

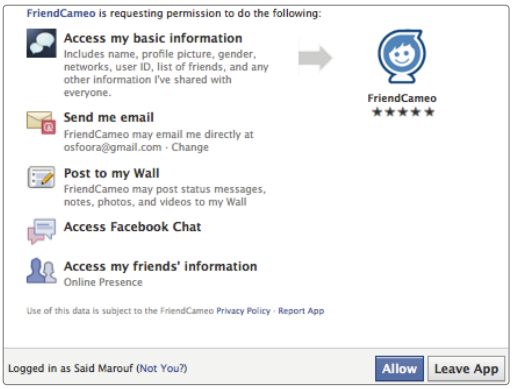
Open standards and third-party software development have long formed a partnership that affords internet users the tools and capabilities to better manage their own identity, privacy, and confidentiality. Seeing a need for users to better know the privacy policies in force for various websites led the World Wide Web Consortium (W3C) to create the Platform for Privacy Preferences (P3P) specification and the corresponding Preference Exchange Language (APPEL) which is in use today by various internet websites to provide, in machine readable format, a policy file specifying that site's particular privacy policies [3].

The OAuth open standard protocol is another example of an available standard created to provide users with the ability to share information and resources with third-party application components of other, more primary, web applications. For example, the OAuth framework might allow for the sharing of photographs from a primary web-based photo sharing website so that a third-party photo printing service may access the permitted photographs [5]. Popular within online social networks, Facebook today represents the largest single OAuth 2.0 implementation permitting a mechanism for third-party web-based applications to access Facebook user identity and privacy information and resources.

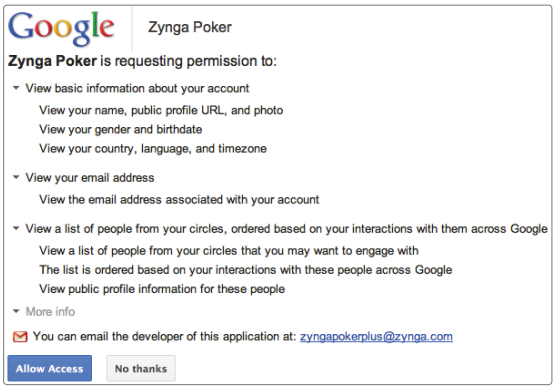
Third-party software developers have led charges to improve user privacy and security, using extensible frameworks available in the Chrome, Firefox, and Safari web browsers. These browser extensions protect users, for example, from unwanted advertisements, malicious software installations, and compromise of user credential data. Indeed, Joshi et al. [22] showcase a browser plug-in that attempts to solve man-in-the-middle attacks prevalent in modern Phishing attacks. While the partnering relationship between standards and browser-based extensions is rich in history and likely to continue, there may exist one gap that needs fulfilling. Appreciating that individual privacy preferences may be just that—individual—how can a single extension reflect the privacy preferences of a unique set of individuals? In this paper, we present a novel browser extension (FBSecure) that implements a proposed recommender-based model, enables users to make important privacy decisions at the time of third-party application installation, and integrates into the existing OAuth 2.0 authorization flow. Recommendations give users confidence in making their decisions, especially that many privacy requests do not clearly convey the accesses requested. The decisions that users make are their own of course, but our algorithm and model provides a mechanism

• *The authors are with the Department of Software and Information Systems, University of North Carolina at Charlotte, 9201 University City Blvd, Charlotte, NC 28223. E-mail: {mshehab, smarouf}@uncc.edu.*

Manuscript received 18 Oct. 2011; revised 9 Feb. 2012; accepted 17 Feb. 2012; published online 13 Mar. 2012.
For information on obtaining reprints of this article, please send e-mail to: tdsc@computer.org, and reference IEEECS Log Number TDSC-2011-10-0238. Digital Object Identifier no. 10.1109/TDSC.2012.34.



(a) Facebook 3rd Party App Permission Request



(b) Google+ 3rd Party App Permission Request

Fig. 1. Social networks application permission requests.

to *inform* them and provide recommendations based on the collaborative decisions (grant/deny) on similar privacy requests within the user's larger social network.

Contributions. Our contributions in this space include

1. A browser extension that intercepts the default OAuth 2.0 request flow, interprets it, and provides the user with a simple interface to make decisions that provide for the protection of private identity attributes *before* application installation.
2. A multicriteria recommender model approach that provides users with recommendations on requested privacy attributes based on the collaborative effort of users who have historically made grant/deny decisions for similarly requested privacy attributes.
3. A recommendation to extend the OAuth 2.0 specification to provide an avenue through which web-browsers (through browser extensions method or otherwise) might assist users in making informed decisions regarding their full privacy attributes before the installation of a third-party application.
4. A user study that shows the results and effectiveness of using our proposed browser extension.

2 Problem Definition

The OAuth framework provides a mechanism for third-party service providers to access end-user resources without releasing the user's access credentials to the service provider. However, specific implementations may not provide the user with the necessary fine-grain access control, nor provide any recommendations on which access control decisions may be the most appropriate. An example we use throughout this paper is one of the free Facebook online video and voice calling applications available through *friendcameo.com*.[1] The FriendCameo Facebook application requests the following extended permissions when a user first installs the application: access to the user's e-mail address, ability to publish status and post messages to the user's wall, the ability to access the Facebook chat application, and the ability to enumerate the online presence status of other users (within the first user's social network).

1. We make no value judgement of the extended permissions requested by the example applications presented in this paper.

It becomes quickly clear that several of the extended permissions, once granted, cannot realistically be revoked. For example, once users provide FriendCameo access to their e-mail addresses, they cannot realistically remove that e-mail address from FriendCameo's servers and databases by preventing further access to the information through Facebook's application privacy settings. We find there are several user attributes that are practically irrevocable once granted, since the attributes are generally immutable (i.e., birthday) or generally change with very little frequency (i.e., hometown locations, religious and political views). (See Fig. 1a) We view the permanent loss of personal attributes as only one part of the problem; should a method be devised to permit users a "last line of defense" against such information loss, how may they know best what decisions to take. Can users benefit from a community of knowledge to better inform their own decision making?

Our proposed approach provides both the aforementioned "last line of defense" mechanism and a recommender model based on the decisions of other users within the community, and the previous decisions of an individual user.

3 Preliminaries

Most of the major online platforms such as Facebook, Google, and Twitter provide an open API which allows third-party applications to directly interact with their platform. APIs provide a mechanism to read, write, or modify user data on such platforms through other third-party applications on behalf of the users themselves. An API comes with a set of methods, each representing a certain user interaction executed through a third-party application. For example, the FriendCameo [12] application is able to post content (e.g., messages, photos) to a user's Facebook feed/wall using Facebook's `/profile_id/feed` API method, where `profile_id` is the targeted Facebook user ID. It is important to note that third-party applications can potentially execute any API call on behalf of a user, relying on the type and scope of permissions granted to these apps. In the previous example, the FriendCameo application could only perform the `profile_id/feed` API call given the user has granted it the `publish_stream` permission. The full set of permissions available to third-party apps are defined by the online platforms, and it is up

to third-party applications to request the proper subset of permissions required. We believe users should have the final decision on which permissions to grant or deny.

3.1 OAuth Standard

With an increasing trend toward offering online services that provide third-party applications the ability to interact through open APIs and access user resources, OAuth was introduced as a secure and efficient mechanism for authorizing third-party applications [28]. Traditional authentication models such as the client-server model require third-party applications to authenticate with online services using the resource owner's private credentials, typically a username and password. This requires users to present their credentials to third-party applications, hence granting them broad access to all their online resources with no restrictions. A user may revoke access from a third-party application by changing her credentials, but doing so subsequently revokes access from *all* third-party applications that continue to use her previous credentials. These issues are amplified given the high number of third-party applications that potentially get access to a user's online resources. OAuth uses a mechanism where the roles of third-party applications and resource owners are separated. It does not require users to share their private credentials with third-party applications, instead it issues a new set of credentials for each application. These new set of credentials are per application, and reflect a unique set of permissions to a user's online resources. In OAuth, these new credentials are represented via an *Access Token*. An Access Token is a string which denotes a certain scope of permissions granted to an application, it also denotes other attributes such as the duration the Access Token is considered valid. We are mainly interested in the scope attribute within an Access Token. Access Tokens are issued by an authorization server after the approval of the resource owner. In this paper, we extend upon this authorization stage of the OAuth 2.0 protocol.

When a third-party application needs to access a user's protected resources, it presents its Access Token to the service provider hosting the resource (e.g., Facebook, Twitter) which in turn verifies the requested access against the scope of permissions denoted by the Token. For example, Alice (resource owner) on Facebook (service provider and resource server) can grant the FriendCameo application (client) access to her e-mail address on her Facebook profile without ever sharing her username and password with FriendCameo. Instead, she authenticates the FriendCameo application with Facebook (authorization server) which in turn provides FriendCameo with a proper Access Token that denotes permission to access Alice's e-mail address.

OAuth provides multiple authorization flows depending on the client (third-party application) type (e.g., webserver, native applications). In this paper, we focus on the *Authorization Code* flow shown in Fig. 2 and detailed in the OAuth 2.0 specification [28]. The authorization code flow is used by third-party applications that are able to interact with a user's web browser, and are able to receive incoming requests via redirection. The authorization flow process consists of three parties: 1) End-user (resource owner) at browser, 2) Client (third-party application), and

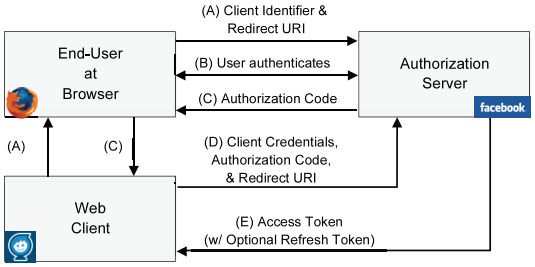


Fig. 2. Authorization code OAuth flow.

3) Authorization server (e.g., Facebook). Our main focus is on steps "(A)" and "(B)" within the authorization code flow [28]. Step "(A)" is where third-party applications initiate the flow by redirecting a user's browser to the authorization server and pass along the requested scope of permissions. In step "(B)," the authorization server authenticates the end user, and establishes her decision on whether to grant or deny the third-party application's access request.

3.2 OAuth and User Privacy

One of the main reasons behind OAuth was to increase user privacy by separating the role of users from that of third-party applications. OAuth uses the concept of Access Tokens, where a token denotes a set of credentials granted to third-party applications by the resource owners [28]. This avoids the need for users to share their private credentials such as their username and password. It also allows users to revoke access to a specific third-party application by revoking its Access Token.

OAuth 2.0 allows third-party applications to request a set of permissions via the `scope` attribute, and for users to grant/deny such requests. If a user grants a third-party application's request, then an Access Token (denoting the `scope`) is issued for that application, hence granting it the scope of permissions requested. The `scope` attribute represents the set of permissions requested by third-party applications, and is our main focus in this paper. In the authorization code, OAuth flow seen in Fig. 2, the `scope` parameter is part of the request URI that is generated by third-party applications (Step "(A)" in Fig. 2). The `scope` is a list of space-delimited strings, each string mapped to a certain permission or access level. For example, the FriendCameo application requests permission to post to a user's Facebook feed/wall, to log in to Facebook chat, to access her e-mail address, and to check her friend's online/offline presence. FriendCameo requests these permissions with a `scope` attribute value of "`publish_stream, xmpp_login, e-mail, friends_online_presence`." The `scope` value becomes part of the OAuth request URI sent to the authorization server (Facebook's OAuth implementation uses commas rather than spaces to separate each requested permission). Step "(B)" of Fig. 2 is where users grant/deny the requested `scope` value.

We propose an extension to the OAuth 2.0 authorization code flow detailed in Section 1.4.1 of the OAuth 2.0 specification [28]. Before users make their decision on the requested `scope` of permissions, we introduce a new level of awareness and control to the user via an in-house developed browser extension.

3.3 Collaborative Filtering

Recommendation systems are systems that try to assist users in evaluating and making decisions on items by providing them opinions and prediction values as a set of recommendations [30]. These set of recommendations are usually based on other people's opinions and the potential relevance of items to a target user. The first recommender system Tapestry [14], followed the approach of "Collaborative Filtering," in which users collaborate toward filtering documents via their individual reactions after reading certain documents. Since then, collaborative filtering has been widely adopted and is accepted as a highly successful technique in recommender systems [24], [27], [25], [33].

In a context of access control and user privacy, items in a collaborative filtering model can be mapped to individual privacy attributes or permissions. Users make decisions on privacy attributes, i.e., grant/deny them to third-party applications. This is similar to other recommendation systems in which users make decisions on items, e.g., to rent a movie or not. Users have their own privacy preferences, but may benefit from the community's collaborative privacy decisions to make their own, especially if they lack the knowledge to make good privacy decisions [20]. The effect of community data on user privacy has been investigated by Besmer et al. [4], who explored the effect of community data on user behavior when configuring access control policies. Their work indicated that community data impacts user behavior, when substantial visual cues were provided. Goecks et al. [13] explored the effects of community data in the domain of firewall policy configuration and web browser cookie management. Their results also indicated that users did utilize community data in making their own decisions.

In this paper, we propose a collaborative filtering model that utilizes community decisions in providing recommendations to users who install third-party applications requesting access to their privacy attributes.

4 Proposed OAuth Flow

We propose an extension to the OAuth 2.0 authorization code flow, by introducing two new modules into the flow: 1) A Permission Guide that guides users through the requested permissions, and shows them a set of recommendations on each of the requested permissions, and 2) a Recommendation Service that retrieves a set of recommendations for the requested permissions following a collaborative filtering model as seen in Section 4.2.

Our OAuth extension focuses on step "(A)" of the authorization code flow in OAuth 2.0 [28]. We revise step "(A)" to become a six stage process as shown in Fig. 3 and explained in the following steps:

A1. The client redirects the browser to the end-user authorization endpoint by initiating a request URI that includes a `scope` parameter.
A2. The Permission Guide extension captures the `scope` value from the request URI and parses the requested permissions. At this step, the extension allows users to choose a subset of the permissions requested.
A3. The Permission Guide extension requests a set of recommendations on the parsed permissions. This is achieved by passing the set of permissions to our Recommendation Service.

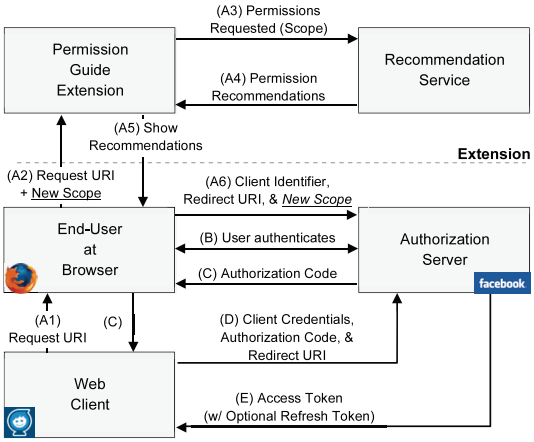


Fig. 3. Proposed OAuth flow.

A4. The Recommendation Service returns a set of recommendations for the permissions requested by the client.
A5. Using the set of returned recommendations, the extension presents the permissions with their respective recommendations in a user-friendly manner.
A6. The Permission Guide extension redirects the end-user's browser to a new request URI with a new scope (`scope'`), assuming the user chooses to modify the requested permissions.

4.1 Permission Guide

The Permission Guide is represented by a browser extension that integrates into the authorization process by capturing the `scope` parameter value within the request URI generated by a third-party application. Once the `scope` is captured, the extension parses the requested permissions and presents them in a user-friendly manner as shown in Fig. 8. A readable label of each requested permission is shown to the end-user, e.g., it shows "Facebook Chat" rather than `xmpp_login`.

The extension also shows users a set of recommendations for the requested permissions. For each permission, there is a thumbs-up and thumbs-down recommendation value. These recommendations represent prediction values that we calculate following our model in Section 4.2. These prediction values represent the likeliness of a user to grant or deny a certain permission based on her previous decisions and on the collaborative decisions of other users. Users who have not made any decisions yet are shown recommendations based on other user decisions.

The extension also allows users to customize the requested permissions by checking or unchecking individual permissions, where a checked permission is one the user wishes to grant to the third-party application and an unchecked permission is one she wishes to deny access to. Once a user decides on the permissions she wishes to grant and deny, she simply needs to click a *Set Permissions* button on the extension (blue button in Fig. 8). This will trigger the extension to generate a new request URI with a new scope `scope'`, and forward the user's browser to this new request URI. `scope'` will always be a subset of the original requested scope, i.e., `scope'` $\subseteq$ `scope`. An example `scope'` for the FriendCameo application could be as follows:

```
scope'=publish_stream
```

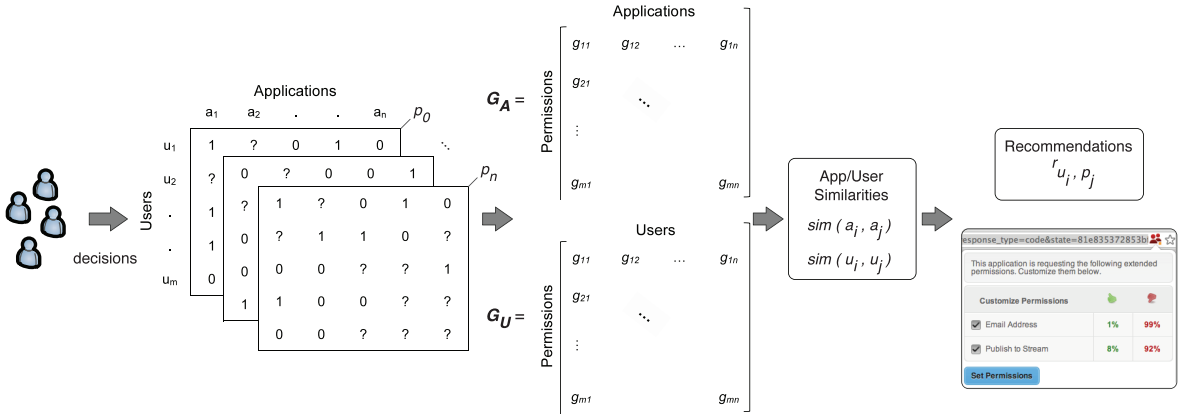


Fig. 4. Collaborative-based model.

reflecting the user's desire to allow FriendCameo to post to her feed/wall, but deny it access to her e-mail, Facebook chat and friend's online/offline presence. Note that using a subset of the permissions requested could potentially hinder the functionality of a third-party application once installed. Investigating such consequence is out of the scope of this paper, but we include it as part of our future work.

Our Permission Guide extension also collects the user's decisions on the requested permissions, hence allows us to generate a data set of decisions to be used in our recommendation model explained in Section 4.2. That is, our Recommendation Service as seen in Fig. 3 will utilize these decisions in making its recommendation predictions. These decisions are uploaded to our servers once a user sets her desired permissions within the extension, i.e., clicks the *Set Permissions* button. The data uploaded to our servers includes: `app_id`, `requested_perms`, `decisions`, `recommendations`, where the `app_id` is the application's unique id which is assigned by the service provider (e.g., Facebook), the `requested_perms` is the scope of permissions requested by the third-party application, the `decisions` are the individual user decisions (grant or deny) on each of the requested permissions, and the `recommendations` are the recommendation values at the time the user made her decisions.

Our goal is to provide a simple user interface for interacting with permission requests, hence increasing user awareness and providing an easy mechanism for guiding users in making their decisions.

4.2 Recommendation Model

We propose a Recommendation Service component that extends upon our Permission Guide extension. Let $\mathcal{A}$, $\mathcal{U}$, and $\mathcal{P}$ represent the set of applications, users, and permissions, respectively. A user $u_i \in \mathcal{U}$ can make a decision $d_i \in \{grant, deny\}$ on a permission $p_j \in \mathcal{P}$ for an application $a_k \in \mathcal{A}$. An application a_k which requests permissions $p_1, \ldots, p_m$ is mapped to a set of decisions $d_1, \ldots, d_m$ made by the user installing a_k.

4.3 Collaborative Filtering

Our model follows the multicriteria recommendation model where user recommendations are calculated per criterion [2], [24]. The model utilizes the set of permissions $\mathcal{P}$ as a set of criteria, i.e., each permission $p_j \in \mathcal{P}$ represents an individual criterion within the model. The multicriteria approach fits our model as decisions are made per permission (criteria) rather than an application as a whole. We model a user's utility for a given application with the user's decisions $d_1, \ldots, d_m$ on each individual permission $p_1, \ldots, p_m$ using Function 1

$$D : Users \times Applications \rightarrow d_1 \times \cdots \times d_m. \tag{1}$$

Function 1 represents a user's overall decision on a certain application via the set of decisions made on each individually requested permission. That is, a user u_i makes a decision d_i on an application a_k with respect to an individual permission. For each permission p_j, there exists a matrix $\mathcal{C}_{p_j}$ representing user decisions on p_j for each application $a_k \in \mathcal{A}$, see Fig. 5. A matrix entry d_i with a value of 1 denotes a user has *granted* a_k the permission p_j, whereas a 0 denotes a *deny*. Entries with "?" values denote the user is yet to make a decision on permission p_j for application a_k. Our model provides recommendations to users that guide them in making these future decisions. Applications that do not request a permission p_j have an empty entry in $\mathcal{C}_{p_j}$ and are handled properly in our implementation.

For example, let $p_1 = birthday$, $p_2 = e-mail$, and $p_3 = location$, where each represents a single criterion within a three-criteria model. Let $u_1 = Alice$ who installed application a_1 that requests access to the permissions *birthday*, *e-mail*, and *location*. As illustrated in Fig. 5, Alice has granted a_1 the permissions *birthday* and *location* ($d_1 = grant, d_3 = grant$), whereas denied *e-mail* ($d_3 = deny$). Alice has yet to make a decision on a_2, i.e., a single decision on each requested

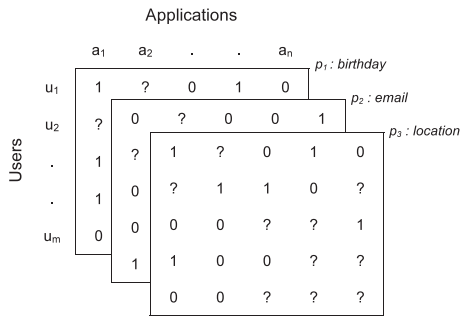


Fig. 5. A three-permission (criteria) model with $\mathcal{C}_{p_1}$, $\mathcal{C}_{p_2}$, and $\mathcal{C}_{p_3}$. User decisions on applications made per permission.

Permissions \ Applications	a_1	a_2	a_3	a_4	a_5
birthday	0.6	0.75	1	0.2	0.3
email	0	0.9	0.25	0.7	0.1
location	1	0.15	0	0.35	0
sms	0	0.4	0	1	0.5
photos	0.2	0	0.6	0.25	0

Fig. 6. Example $G_A(j,k)$ values.

permission $\in \{birthday, e-mail, location\}$. Our proposed model utilizes the set of decisions for each $\mathcal{C}_{p_j}$, hence providing a recommendation that fits each criterion.

Fig. 4 illustrates our overall collaborative model. The model relies on decisions made by the community users, and utilizes them in building the multicriteria matrices $\mathcal{C}$ for each permission. By utilizing the $\mathcal{C}$ matrices, we generate two probability matrices, G_A and G_U, as seen in Fig. 4. G_A is app based, whereas G_U is user based. G_A captures the probability of a certain application being granted a certain permission, whereas G_U captures the probability of a certain user granting a certain permission.

Fig. 6 shows an example G_A matrix, with a set of applications (a_1, a_2, a_3, a_4, a_5), permissions (`birthday`, `e-mail`, `location`, `sms`, `photos`) and their corresponding $G_A(j,k)$ values. For example, $G_A(location, a_2) = 0.15$, denotes a low probability of the permission `location` being granted to application a_2 by users who installed a_2. Our proposed collaborative model adopts an item-based and user-based collaborative filtering process. In our model, items are applications; hence, we refer to item-based filtering as application-based filtering. User-based filtering utilizes the user-based probability values of G_U, whereas application-based filtering utilizes the app-based probabilities of G_A as seen in Fig. 4.

4.3.1 Application-Based Filtering

Our application-based filtering process relies on the app-based probability values of G_A shown in Fig. 4. Each entry $G_A(j,k)$ in G_A represents the overall probability of permission p_j being granted to application a_k.

To generate recommendations on the requested permissions, we first detect the nearest neighbors for the target application requesting the permissions. The nearest neighbors in app-based filtering are the applications most similar to the target application. Collaborative filtering algorithms have mainly been based on one of two popular similarity measures namely the Pearson Correlation and Cosine similarity [18], [27]. We measure similarities between applications using the G_A values, and by calculating the Pearson correlation values between them. Equation (2) represents our application-based similarity measure, which is the Pearson correlation value between applications a_i and a_j, where $\mathcal{P}$ is the set of all permissions in our system and $\overline{G_A}(a_i)$ is the average probability for application a_i being granted a permission in $\mathcal{P}$

$$sim(i,j) = \frac{\sum_{\forall p \in \mathcal{P}} (G_A(p,i) - \overline{G_A}(a_i))(G_A(p,j) - \overline{G_A}(a_j))}{\sqrt{\sum_{\forall p \in \mathcal{P}} (G_A(p,i) - \overline{G_A}(a_i))^2 \sum_{\forall p \in \mathcal{P}} (G_A(p,j) - \overline{G_A}(a_j))^2}}. \quad (2)$$

Applications that don't request a certain permission p_j have a $G_A(j,i)$ of zero. Applications which are similar and highly correlated are those which request a similar set of permissions, and have similar $G_A(j,i)$ values for each of their requested permissions. For example, if both applications a_1 and a_2 requested the same set of permissions $\{p1, p2\}$, and they have a $G_A(p_1,a_1) = G_A(p_1,a_2)$ and a $G_A(p_2,a_1) = G_A(p_2,a_2)$, then a_1 and a_2 are considered highly correlated and their application-similarity value $sim(i,j)$ will be close to 1. When predicting recommendation values for permissions of application a_i, we make sure they are based on a_i's *nearest neighbors*, that is, the set of applications where $sim(a_i,a_j)$ is highest. With application-based filtering, users collaborate toward increasing or decreasing the $G_A(j,k)$ values, hence filtering applications according to the willingness of users to grant them certain permissions.

4.3.2 User-Based Filtering

User-based filtering relies on the G_U values, where each entry $G_U(j,k)$ in G_U represents the overall probability of permission p_j being granted by a focus user u_k. Permission recommendations in this case are based on the focus user's nearest neighbors, that is, the users most similar to the focus user. Similar to application-based filtering, we use the Pearson correlation to measure similarities between users. Equation (3) represents our user-based similarity measure, which in terms is the Pearson correlation value between users u_i and u_j, where $\overline{G_U}(u_i)$ is the average probability of user u_i granting a permission in $\mathcal{P}$

$$sim(i,j) = \frac{\sum_{\forall p \in \mathcal{P}} (G_U(p,i) - \overline{G_U}(u_i))(G_U(p,j) - \overline{G_U}(u_j))}{\sqrt{\sum_{\forall p \in \mathcal{P}} (G_U(p,i) - \overline{G_U}(u_i))^2 \sum_{\forall p \in \mathcal{P}} (G_U(p,j) - \overline{G_U}(u_j))^2}}. \quad (3)$$

With user-based filtering, a focus user u_i is given recommendations based on those users most similar to him/her. Users with more similar probabilities of granting a certain permission will be more similar, hence, potentially reflect a similar willingness to grant/deny a certain permission.

We use both application-based and user-based filtering to calculate a recommendation value on permissions requested by application a_i on behalf of user u_i.

4.4 Prediction Model

When a user u_i, say Alice, wants to install application a_k, we calculate a set $\mathcal{R}_k$, where $r_{i,j} \in \mathcal{R}_k$ is a prediction value for permission p_j requested by a_k. $r_{i,j} \in \mathcal{R}_k$ is a prediction of how likely Alice would be willing to grant p_j to a_k.

The recommendation value $r_{i,j}$ is based on either our app-based filtering or user-based filtering approaches. That is, the recommendations are either based on a_i's nearest neighbors (most similar applications) or u_i's nearest neighbors (most similar users). Equations (4) and (5) show

the recommendation value for app-based and user-based filtering, respectively. Note that we calculate $r_{i,j}$ for each p_j requested by an application a_k

$$r_{i,j} = \overline{G_A}(p_j) + \frac{\sum_{a \in \mathcal{N}} sim(a_k, a) * d_{j,a}}{\sum_{a \in \mathcal{N}} |sim(a_k, a)|} \tag{4}$$

$$r_{i,j} = \overline{G_U}(p_j) + \frac{\sum_{u \in \mathcal{N}} sim(u_i, u) * d_{j,a_k}}{\sum_{u \in \mathcal{N}} |sim(u_i, u)|}. \tag{5}$$

In (4), $\overline{G_A}(p_j))$ reflects the average probability that permission p_j is granted over all applications in $\mathcal{A}$, and is easily calculated via its corresponding row in the G_A matrix. Similarly, in (5), $\overline{G_U}(p_j)$ represents the average probability that permission p_j is granted over all users in $\mathcal{U}$, and is calculated via its corresponding row in the G_U matrix. Note that both $\overline{G_A}(p_j))$ and $\overline{G_U}(p_j)$ are driven by all users within our system. In both equations, $\mathcal{N}$ represents the target application's nearest neighbors and the focus user's nearest neighbors, respectively. The size of $\mathcal{N}$ depends on the similarity measures used, and can be adjusted to follow a preset threshold within the implementation, e.g., only include neighbors with a similarity above 0.8.

Finally, $d_{j,a}$ in (4) represents u_i's (focus user) previous decisions on permission p_j for each application $a \in \mathcal{N}$. In (5), d_{j,a_k} is a neighboring user's decision on p_j for the focus application a_k. Note that the $sim(u_i, u)$ value will either increase or decrease the effect of a neighboring user's decision, based on how similar the neighboring user is to the focus user. Both $d_{j,a}$ and d_{j,a_i} are captured via the $\mathcal{C}_{p_j}$ matrix explained earlier (see Fig. 5).

Notice that the prediction values calculated are based on a user's previous decisions and on the decisions of other users, hence capturing the essence of collaborative filtering. In cases of insufficient data, prediction models could refrain from generating predictions, or utilize collaborative filtering systems based on probabilistic, hybrid, or clustering approaches for generating predictions. We decided not to provide predictions in such cases.

4.4.1 Category-Based Predictions

To further enhance the results of our recommendation predictions, we propose a category-based model that takes into consideration an application's category. Example application categories include Games, Utilities, Entertainment, etc. Categories can increase the precision of our predictions especially for applications that request similar permissions for different purposes. For example, two applications might request access to a user's e-mail address, where the first application is a game and the second is a task manager. In this example scenario, a user's e-mail could be used for different purposes, i.e., a task manager could use it for sending reminder e-mails, whereas a game could use it to send promotions for other games. A user would probably be more willing to grant e-mail permission to the task manager as it could be of more benefit to the user. Granting or denying a certain permission will be driven by the user's perception of the requested permission. We believe that similar permissions requested by apps within the same category will be perceived similarly by users. Hence, by providing recommendation predictions based on application categories, we can reflect more precise user perceptions within our recommendations.

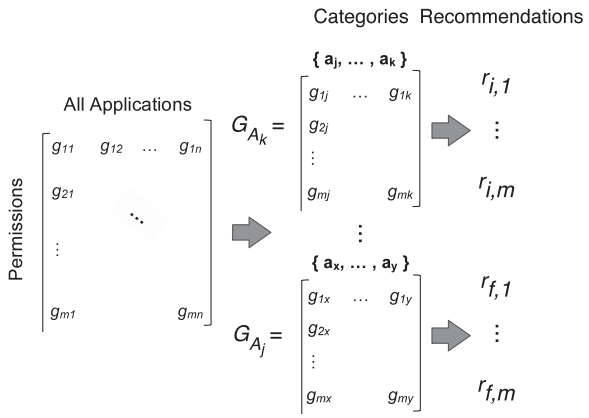


Fig. 7. Application category probability matrices. Recommendations per app category.

When generating category-based predictions, we follow a modified version of our application-based filtering model for calculating similarities. To calculate the set of nearest neighbors for a certain application a_i, we only consider other applications that fall into the same category as a_i. Fig. 7 shows two probability matrices G_{A_k} and G_{A_j}, which are extracted from the overall G_A matrix explained previously. G_{A_k} and G_{A_j} represent the permission probabilities for applications within the categories k and j, respectively. Let $\mathcal{A}_k \subseteq \mathcal{A}$ be the set of applications that belong to category k, and $\mathcal{N}_i$ be a_i's nearest neighbors where $\mathcal{N}_i \subseteq \mathcal{A}_k$. Note that a_i's nearest neighbors can be found by calculating the similarities between a_i and applications within $\mathcal{A}_k$ rather than all applications in $\mathcal{A}$. For example, in Fig. 7, the nearest neighbors for a_y are found among the set of apps $\{a_x \ldots a_y\}$, and the similarities are calculated using G_{A_j}. For application $a_i \in \mathcal{A}$ that belongs to category k, we calculate recommendation predictions following:

$$r_{i,j} = \overline{G_{A_k}}(p_j) + \frac{\sum_{a \in \mathcal{N}_i} sim(a_i, a) * d_{j,a}}{\sum_{a \in \mathcal{N}_i} |sim(a_i, a)|}, \tag{6}$$

where $\overline{G_{A_k}}(p_j)$ reflects the average probability that permission p_j is granted over applications in $\mathcal{A}_k$, i.e., apps that fall within a_i's category. Category-based predictions are more efficient in that they do not rely on all applications within our system, but rather on a smaller subset of categorized applications. This allows for faster prediction calculations, in addition to the potentially more precise recommendations.

5 Benefits and Attacks

In presenting our browser extension, we must also acknowledge the various attacks against which it might be purported. We believe the most likely attack scenarios center around abusing the extension to manipulate the recommender model into recommending decisions that most favor an attacker. These attacks might take the form of

1. Frequent use of the extension for the same application. Attackers may use the extension to load the same application repeatedly in the hopes to increase the frequency of their own decision in the underlying database. For example, an attacker may want to direct users to reveal their e-mail address and so may attempt to load the OAuth request

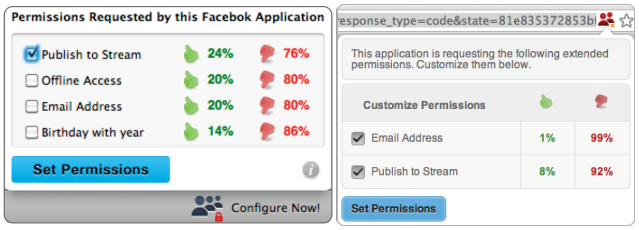
repeatedly, choosing that option. We can mitigate this risk considerably by linking the user's request to their Facebook user id (available to our extension through browser cookies) and ensuring that only the most recent permission decisions are stored in the database. As part of our future work, we would like to be able to analyze decisions over time. To do so, we need to hold onto user decisions over time. To defend against attacks in this case, we override the latest user decision and only consider the latest decision as part of our recommendation predictions.

2. *Decoding and manipulation of the API.* Attackers may decode the extension and exploit its internal programming interface to make direct calls to the underlying web service in an attempt to inflate user statistics with an eye to influencing decisions. Our previous mitigation method will be useless here since any information sent to the web service may be captured and replayed, perhaps varying slightly the unique user information in an appearance of legitimacy. Further complicating any countermeasures is the fact that the entire code resides on the client computer and can reasonably be suspect to successful reverse engineering. In this case, we propose to use out-of-band methods to attempt to deter and detect fraud. We can use the source IP address as one method to further link an individual web service call to a unique user. This has an admittedly loss of fidelity when user requests are routed through proxies (such as those found in institutional use, or at very large internet service providers). Combining an IP address with a minimum time interval between requests may provide us adequate protection in this regard. Limiting unique web-service calls to calls within a 15 minute window from the same source IP address may serve as a sufficient preventative control; postrequest log file analysis may further provide some detective measures toward preventing fraud. Our web service must also be resilient against known web attacks and our servers undergo web application and server scanning to prevent the most common and known attack vectors such as buffer overflow, SQL injection, and cross-site scripting attacks. We also note that we make sure that users are aware of the data collected during our user study.

6 Experiments

We evaluate our proposed OAuth 2.0 extension using Facebook as our target platform. Facebook is an ideal target given its large user base of over 800 million users, and its extensive application directory of over seven million third-party applications [8]. Facebook is also one of the major platforms to adopt the OAuth 2.0 protocol, which makes it a good fit for our evaluation process. The proposed extension is not limited to Facebook and can be extended to other OAuth 2.0 platforms. To evaluate our proposed OAuth 2.0 extension, we implemented two main components: a Permission Guide, and a Recommendation Service.

Permission guide. Our proposed Permission Guide in Section 4.1 was implemented as a browser extension for both Firefox and Chrome browsers, using a combination of Mozilla's XML User Interface Language, the Google Chrome browser APIs and Javascript. Fig. 8 shows the extension user interface for both Firefox and Chrome. Javascript was used to interact with our back-end recommendation service API. The extension was tested on the latest Firefox and Chrome browsers on Mac OS X 10.6/10.7, Linux CentOS and Windows (Vista, 7) machines.



FireFox Extension UI Chrome Extension UI

Fig. 8. Extension UI, Firefox, and Chrome.

Once installed, the extension resides within the user's browser and begins monitoring, waiting for a Facebook application installation process to commence. The extension does not otherwise interfere with a user's browsing experience. Once a Facebook application installation process is detected, the extension is activated and presented to the user.

An installation process is detected by parsing the URLs a user visits and searching for a *Permission Request*. A *Permission Request* for Facebook applications can be identified by locating the substrings *permissions.request* and either of *facebook.com/connect/uiserver* or *facebook.com/dialog/permissions.request*. If a request is detected, the extension looks for the type of request issued, i.e., Basic permission versus Extended permission access. A basic permission access request is identified by a missing or empty `scope` attribute within the URL. Otherwise, if the *scope* attribute is located, the extension recognizes that an extended permission access request is in progress.

Recommendation service. The service is a PHP-based solution running on Apache 2.2.14 with MySQL 5.1.5 as the data store solution. We run the service on a desktop machine running Linux CentOS, with 2 GB RAM and a 2.0 GHz Intel Xeon CPU. The recommendation service applies the recommendation-based schema explained in Section 4.2 by providing two private API methods which are used by our extension. The first API method is the `getRecommendations` method which accepts an `app_id` and a set of requested permissions. It then returns a set of recommendations in a JSON format which maps a recommendation value to each permission. The second API method provided is the `postDecisions` method which is invoked by our extension when a user makes her decision on the requested permissions. This API method takes an `app_id`, a set of requested permissions, a set of user decisions on these permissions, and the set of recommendation values displayed at decision time. These values are stored onto our recommendation back-end server and used later in our recommendation-based schema.

For our evaluation purposes, we are primarily focused on extended permission requests because those are the permissions which are customizable by users on the targeted platform (Facebook). For basic permission requests, our extension notifies users that basic access is requested, and no customization is possible. Whereas for extended permission requests our extension performs the following:

1. Extracts the permissions requested by parsing the `scope` value from within the request URI. For Facebook, the `scope` value is a list of comma-delimited strings, each string representing a certain requested permission.

2. Asynchronously retrieves recommendations for the set of requested permissions by calling our API method `getRecommendations`. Once the recommendations are retrieved, the extension UI is updated properly.
3. Dynamically generates the user interface to be shown to the user based on the requested permissions and their respective recommendation values. Fig. 8 shows an example interface for

`scope = publish_stream, offline_access, e-mail, birthday.`

Once the user makes a decision on the permissions she would like to grant/deny by clicking the "Set Permissions" button, the extension will perform two actions: 1) Invoke our `postDecisions` API method passing along the user decisions. 2) Generate a new `scope` value using the permissions granted by the user. Using this new scope `scope'` the user is then redirected to a customized application request URI, resulting in a new Facebook application permission request page. At this point, the user has defended herself against unnecessary application accesses. Note that our approach prevents an application from acquiring permissions before its actual installation. The current approach by Facebook allows the removal of permissions only after applications are installed, which is realistically not sufficient because applications have already acquired access to the data.

6.1 User Study

To evaluate our proposed framework, we perform a user study on our browser extension FBSecure. The study's main research questions were: *1) Do permission recommendations (positive/negative) affect the user's willingness to allow/deny permissions requested by third-party applications?* and *2) Are users more willing to share their friends' privacy attributes in comparison to their own?* We use statistical measures to evaluate the success of our proposed framework as discussed in Section 6.1.2.

6.1.1 Methodology

Our proposed browser extension is hosted under the name of FBSecure on the Mozilla Add-Ons website (Firefox version) and, the Google Chrome web store website (Chrome version). In addition, it was posted on our lab website (http://liisp.uncc.edu/fbs). Twitter was also used as a means of recruiting participants for this study which was approved by UNC Charlotte IRB (Protocol# 11-05-24). FBSecure was installed by over 3,528 Facebook users who installed over 1,561 unique Facebook applications. The results summarized in this section are based on the population of users who installed our browser extension, use Facebook, and sought out privacy extensions. This user sample is mainly biased toward privacy aware users, but also includes regular users recruited via Twitter, whom did not specifically seek out privacy extensions.

6.1.2 Study Results

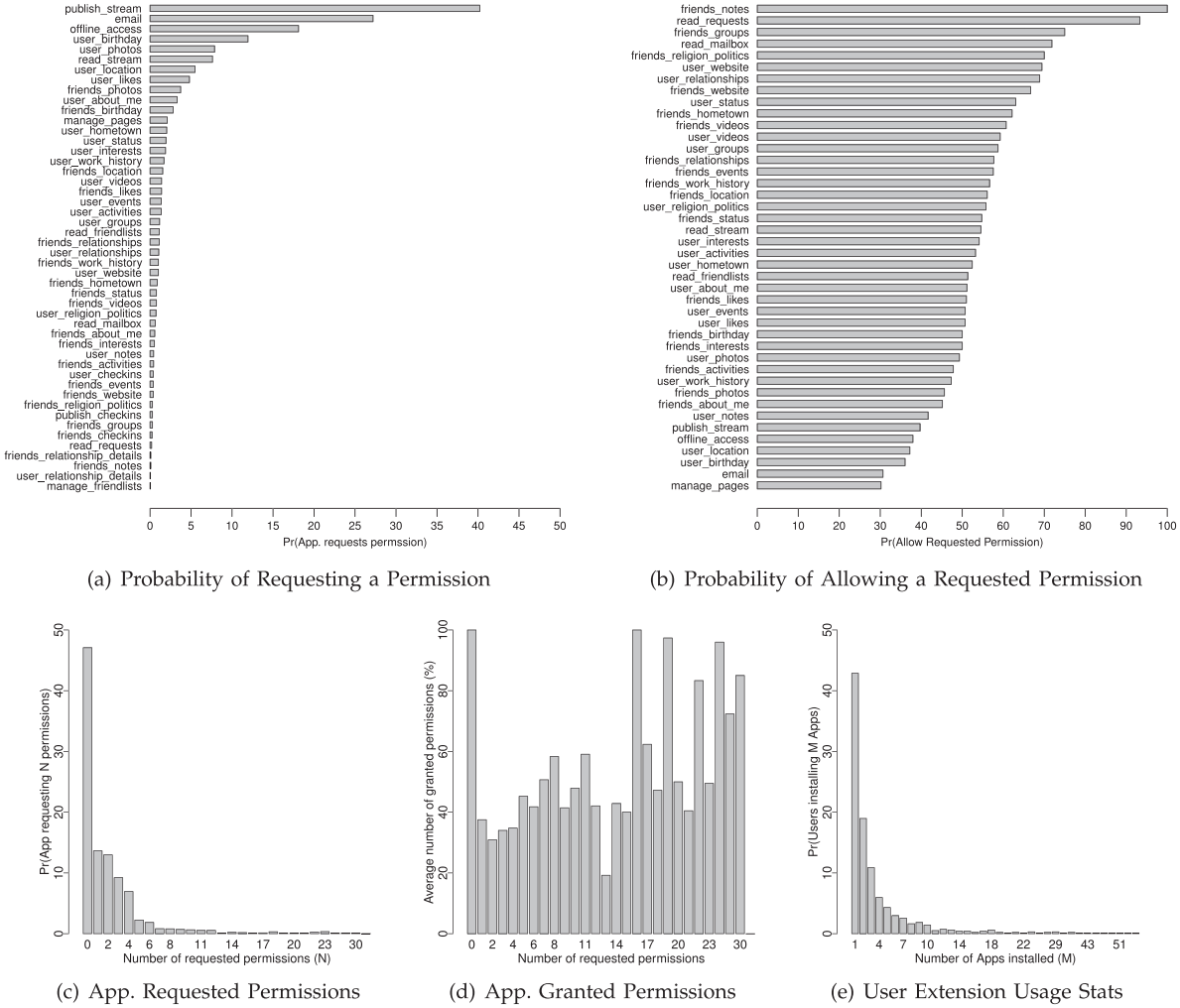
We gathered over 7,200 user decisions on 56 different Facebook extended permissions. We evaluate our recommendation model based on the user decisions collected during the usage of the extension. For every application permission request, our extension enabled the collection of the details of the requested permission, the generated recommendation, and the user-selected permission settings. Fig. 9a shows the probability of applications requesting different permissions, for example, we found that the most popular requested permission is the `publish_stream` permission, which enables apps to post messages on a user's wall, and is requested by 42 percent of the Facebook apps. Other popular permissions include `e-mail`, `offline_access`, and `user_birthday`.

Over all our user population, Fig. 9b shows how likely users were willing to grant different permissions. Our results show that users have varying willingness toward different permissions, for example, the likelihood of a user giving an application access to his e-mail is only 31 percent, while users are more likely to share their status (65 percent) with apps. Note that some permissions requested give applications access to user's friends' information, for example, `friends_location` permission. To investigate the permissions that users are more willing to grant on their friends' data compared to their own data, we conducted a t-test on the likely of allow statistic collected when users are asked for permission to access both their data and their friends' data. With a significance level of five percent, Fig. 10 shows the permissions for the hypothesis that users are more willing to share their friends data are accepted. For example, it is statistically significant that users are more willing to share with apps their friends' birthday compared to their birthday.

Fig. 9c summarizes the distribution of the number of permissions requested by applications, with an average of 3.1 permissions requested per application. Fig. 9d shows the average number of granted permissions for apps requesting permissions, and it can be noted that on average applications are granted around 44.7 percent of the permissions that are requested. Fig. 9e shows the distribution of number of applications by users who installed the extension, on average the extension was used to install 5.2 applications.

The extension provides users with recommendations for each of the application requested permissions. The recommendation is presented to the user as thumbs up and thumbs down with their associated recommendation values based on the recommender models presented in previous sections. We are interested in evaluating whether the recommender system properly predicts the user's decision. Also, we are interested in evaluating what is the lowest (highest) recommendation value that will influence users into granting (denying) a requested permission, we refer to this value as the threshold T where users said to be encouraged to grant the permission if the recommendation is higher than T and to deny otherwise. In this case, we have four possible outcomes for the recommended and decided value, see Fig. 11.

In the literature, there are several proposed metrics for evaluating recommender system performance, we focus on the most adopted metrics in the literature which are based on three measures namely accuracy, precision, and recall [19]. Accuracy of the recommender system is the degree of closeness of the recommender system to the actual decision taken by the user, which is calculated as $\frac{TP+TN}{TP+TN+FP+FN}$. The precision or the repeatability of the recommender system is a measure of the degree to which repeated recommendations under the similar conditions generate the same results,



(a) Probability of Requesting a Permission

(b) Probability of Allowing a Requested Permission

(c) App. Requested Permissions

(d) App. Granted Permissions

(e) User Extension Usage Stats

Fig. 9. Application and user permission probability profiles.

which is computed as $\frac{TP}{TP+FP}$. The recall or sensitivity is a measure of the ability of the recommender system to select instances of either to recommend or not, which is computed as, $\frac{TP}{TP+FN}$. Fig. 12 shows the accuracy, precision, and recall calculated for different threshold values. The experiments were conducted to evaluate the proposed application-based, user-based, and category-based recommendation models. The application- and category-based approaches maintained an accuracy of over 90 percent. The category-based approach provided the highest accuracy, this is due to the refined application similarity value as apps in a given category provide a better context for providing recommendations for apps from the same category. The precision and recall are inversely proportional with a break even region around the threshold value of 45 percent, which could explain that the recommendation value of 45 percent or higher is an indication that the system is recommending to grant the requested permission, and lower than 45 percent is recommending to deny the permission. Also, note that the system achieves a precision and recall values of 85-92 percent and 75-85 percent around this threshold.

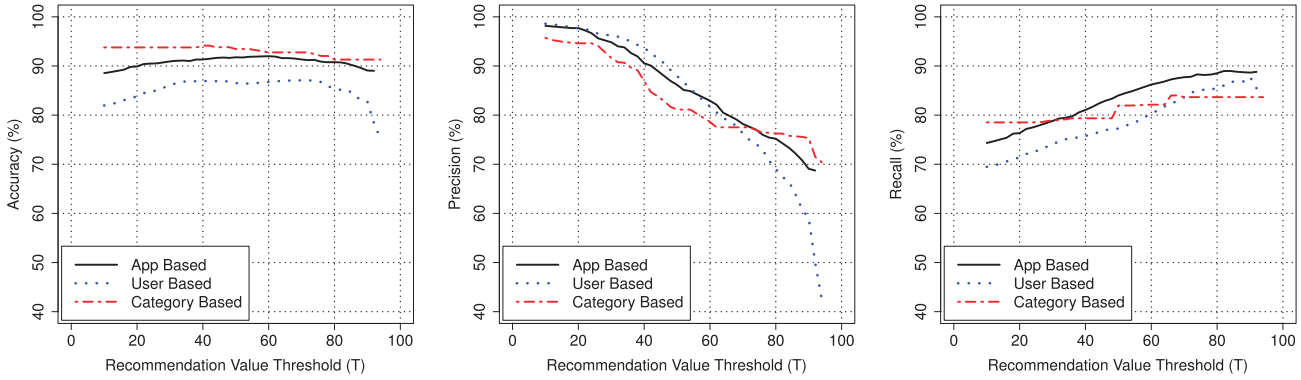
In addition to investigating the accuracy, precision, and recall measures, we further investigated the causality of our recommendation scheme. That is, are users less likely to grant permissions when using the recommendation-based scheme? To investigate, our browser extension was designed to accommodate two groups of users. The first group (G1) are users who were not shown the recommendation values (see Fig. 13a). The second group (G2) are users who were shown the recommendation values generated by the recommendation system (see Fig. 13b). The extension randomly selected users who belonged in each of the groups. For each group, we recorded the users' *openness*, which is the percentage of

Attribute	User (μ, σ)	Friend (μ, σ)	p-value
notes	(0.42, 0.50)	(0.98, 0.21)	0.0019
birthday	(0.38, 0.45)	(0.48, 0.46)	0.0123
location	(0.38, 0.44)	(0.57, 0.45)	0.0144
groups	(0.57, 0.47)	(0.75, 0.42)	0.0253
work_history	(0.45, 0.43)	(0.58, 0.44)	0.0313
religion_politics	(0.56, 0.48)	(0.71, 0.50)	0.0377
online_presence	(0.38, 0.42)	(0.56, 0.49)	0.0456
events	(0.51, 0.51)	(0.60, 0.58)	0.0475
videos	(0.58, 0.41)	(0.61, 0.44)	0.0491

Fig. 10. T-test user and friend permissions.

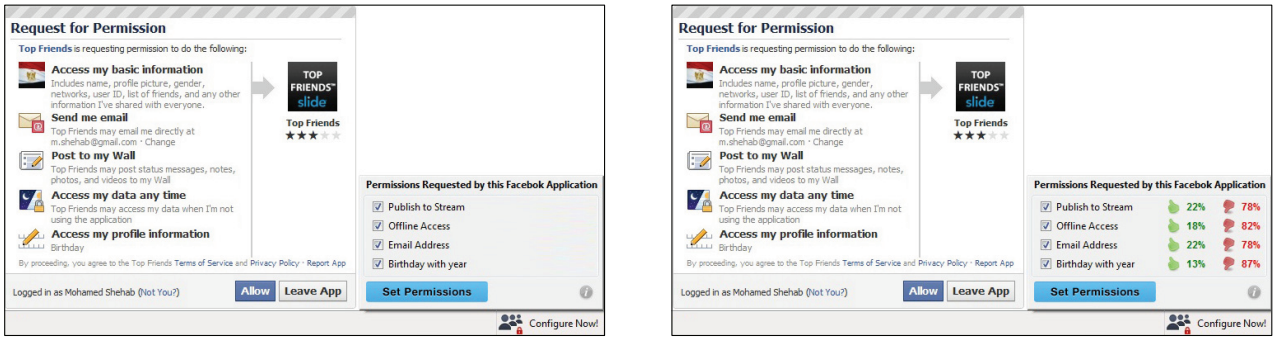
	Recommended	Not Recommended
Used	True Positive (TP)	False Negative (FN)
Not Used	False Positive (FP)	True Negative (TN)

Fig. 11. Classification based on user decisions.



(a) Accuracy (b) Precision (c) Recall

Fig. 12. Recommendation system accuracy, precision, and recall evaluation.



(a) Group (G1) no recommendation shown (b) Group (G2) recommendation shown

Fig. 13. Recommendation experimental setup.

granted permissions for each application installed. The average user openness of G1 and G2 were 66.5 and 30.7 percent, respectively, which indicates that users who were not presented with the recommendation were more likely to grant permissions to applications. To compare the two groups, we performed a T-test of the hypothesis to investigate the following question, "on average, are users in G2 less open than users in G1?" Using the collected data, with a significance level of five percent this hypothesis was accepted (P-Value of 0.0001). These results show that the users who were presented with the recommendation values were less open to granting permissions to applications. The results presented in this experiment are based on the average openness values calculated over all installed apps in both groups. Fig. 14 shows the expected openness for the two groups for specific permissions for which the hypothesis was accepted.

7 Related Work

Developing usable tools that provide fine-grained control over user private data is an emerging problem in online platforms especially within social networks (e.g., Facebook, Google+) [15], [1], [6], [17]. Studies such as the one by Acquisti and Gross [16], [1] indicate user concern over their privacy on social networks while most users did not apply strict privacy settings on their online social profiles. This was mostly due to the lack or poor understanding of what privacy controls are available to them. Our work in this paper focuses on providing a usable tool via a browser extension which allows for users to easily understand and customize their privacy settings at application installation time. We also increase user awareness by providing them a set of recommendations on the requested privacy attributes.

Using browser-based plug-ins and extensions is another widely used approach in aiding information privacy. Jenkin and Dymond [21] originally identified this approach to address the problem of "an information provider wanting to serve secrets embedded within regular web-pages to authorized users." The authors' original italicized elements hold true for our model, substituting users for information providers, identity attributes for secrets, and authorized third-party applications for authorized users. Today, the "open source" model is routinely used to enable

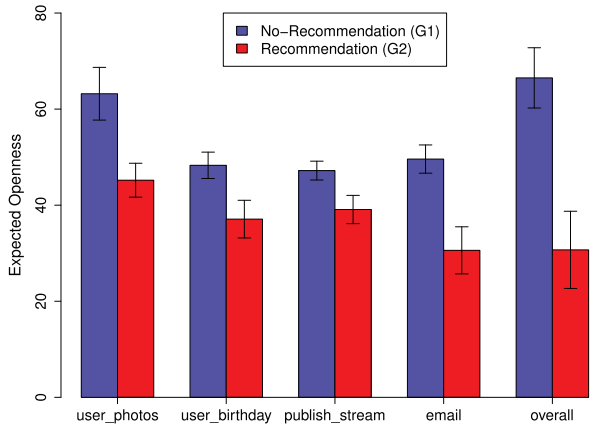


Fig. 14. Groups (G1) and (G2) expected openness.

community-contributed plug-ins for web browsers that aid in some aspect of security; as one example, the Mozilla Firefox browser boasts over 500 security and privacy plug-ins available at their add-ons website.

Felt and Evans [10] detail a novel solution for protecting privacy within social networking platforms through the use of an application programming interface to which independent application owners would agree to adhere to. Our approach requires no such agreement and, barring a wholesale adoption of a privacy proxy such as the one which Felt proposes, still enables the user to protect their information attributes. We achieve this by utilizing the already popular OAuth 2.0 authorization flow and providing a seamless experience to users for customizing and protecting their private information attributes. Recently, Felt et al. reviewed the permissions requested by current applications [11]. While some of their findings apply to the context of Android applications, they confirm our claim that up-front permission requirements for installation may help APIs achieve their full potential in a secure fashion, while still be useful for end-users.

Fang and LeFevre's work asserts the value in providing highly accurate privacy settings with reduced user input [9]. Using real user input, they infer a set of privacy preferences using a machine learning approach. While the authors' study is based on real users, they do not provide a technique that applies the inferred privacy settings onto a user's real online profile. Our implemented browser extension is not only based on real user data, but also capable of applying a user's desired privacy decisions to their real online profile. They also focus on privacy settings related to a user's friends network, whereas our work focuses on third-party applications which we believe represent a bigger threat to user privacy. We also note that using a machine learning technique is not ideal in situations where instant privacy suggestions are required, which is the case when installing third-party applications within social networks such as Facebook.

Besmer et al. [4] demonstrated in their research the value of social navigation cues in prompting users to make informed privacy decisions, where that research was not concerned with the type of data and arbitrarily assigned a recommended positive or negative cue for each item, our research is very specifically tied to data types and our recommender model provides cues that are based on real user privacy decisions. Our work is also significantly different in that we provide a real-life user study through a real-world implementation of a browser extension that integrates with a user's actual Facebook profile, where the research by Besmer et al. was experimental. We also note that our browser extension applies a user's privacy decisions onto their real privacy settings within Facebook.

While much has been researched about the privacy impacts of recommender systems themselves [31], [29], [7], little research appears to be available for the use of recommender systems in aiding privacy and security systems. One notable exception is in the research of Kelly et al. [23] where the authors demonstrated the benefit of combining collaboration among a user population in the suggestion of an individual user's privacy policy. They also propose an incremental model for optimizing a user's policy over time. We find this approach not optimal when dealing with third-party applications, that once installed, can harvest a user's private social network data. Optimal and instant privacy protection should be provided to users at installation time, which we achieve through our browser extension. Liu and Terzi offer a framework for deriving a "privacy score" to inform the user of the potential risk to their privacy created by their activities within the social network [26]. However, such research does not account for the discrete control over attributes which our research enables.

Goecks et al. [13] explore the effects of community data in the domains of firewall policy configuration and web browser cookie management. Their research indicates that users utilized community data in making their privacy decisions. They also investigate the effects of *informational cascades* and the possible misuse of community data within social navigation systems. They present two approaches for mitigating the effects, and we believe these can complement our work.

Shehab et al. [32] proposed an access control framework that allows users to specify the data attributes to share with applications and the degree of specificity. Their framework requires many changes to existing authorization models and requires developers to go through a cumbersome deployment process. Our proposed framework integrates seamlessly into existing authorization models, and requires no additional effort from developers.

8 Conclusion and Future Work

Usable privacy configuration tools are essential in providing user privacy and protecting their data from third-party applications in social networks. We proposed an extension to the authorization code flow of OAuth 2.0 and implemented a browser extension that integrates into the existing OAuth flow, and allows users to easily configure their privacy settings for applications at installation time. We also proposed a multicriteria recommendation model which adopts three collaborative filtering techniques: app-based, user-based, and category-based, each incorporating the decisions of the community and previous decisions of an individual user. Based on this model, our browser extension provides users with recommendations on permissions requested by applications. We successfully demonstrate that our extension, combined with our multicriteria recommendation model leads to the preservation of irrevocable, immutable private identity attributes and the preventing of their uninformed disclosure during application installation. Among popularly requested permissions, individuals when given the choice are more likely to deny the requested permission. We demonstrate the effectiveness of the recommendations through a causal group of users who were not shown any recommendations, and we found them to be more willing to grant permissions to third-party applications than those who were provided with recommendations. In the future, we will investigate application permission evolution over time and address possible application misconfigurations due to insufficient permissions. We also plan on investigating probabilistic and hybrid collaborative filtering systems for providing better predictions in cases of sparse user decision data. We'd also like to investigate the benefits of providing additional information (e.g., population age distribution) to users when making their privacy decisions. Additionally, we would like to investigate the merits of our approach on other platforms, e.g., mobile platforms.

Acknowledgments

This work was funded by the US National Science Foundation (NSF-CNS-0831360, NSF-CNS-1117411) and Google Research Award.

References

[1] A. Acquisti and R. Gross, "Imagined Communities: Awareness, Information Sharing, and Privacy on the Facebook," *Proc. Int'l Workshop Privacy Enhancing Technologies,* pp. 36-58, 2006.

[2] G. Adomavicius and Y. Kwon, "Multi-Criteria Recommender Systems," *Recommender Systems Handbook: A Complete Guide for Research Scientists and Practitioners,* Springer, 2010.

[3] G.-J. Ahn, M. Ko, and M. Shehab, "Privacy-Enhanced User-Centric Identity Management," *Proc. IEEE Int'l Conf. Comm. (ICC),* pp. 1-5, 2009.

[4] A. Besmer, J. Watson, and H.R. Lipford, "The Impact of Social Navigation on Privacy Policy Configuration," *Proc. Sixth Symp. Usable Privacy and Security (SOUPS '10),* July 2010.

[5] W. Bin, H.H. Yuan, L.X. Xi, and X.J. Min, "Open Identity Management Framework for Saas Ecosystem," *Proc. IEEE Int'l Conf. e-Business Eng. (ICEBE '09),* pp. 512- 517, 2009.

[6] D. Carrie and E. Gates, "Access Control Requirements for Web 2.0 Security and Privacy," *Proc. Workshop Web 2.0 Security & Privacy (W2SP '07),* 2007.

[7] S. Chen and M.-A. Williams, "Towards a Comprehensive Requirements Architecture for Privacy-Aware Social Recommender Systems," *APCCM '10: Proc. Seventh Asia-Pacific Conf. Conceptual Modelling,* pp. 33-42, 2010.

[8] Facebook, Facebook Press Room, http://www.facebook.com/press/info.php?statistics, 2011.

[9] L. Fang and K. LeFevre, "Privacy Wizards for Social Networking Sites," *Proc. Int'l Conf. World Wide Web (WWW),* M. Rappa, P. Jones, J. Freire, and S. Chakrabarti, ed., pp. 351-360, 2010.

[10] A. Felt and D. Evans, "Privacy Protection for Social Networking Platforms," *Proc. Workshop Web 2.0 Security and Privacy,* 2008.

[11] A.P. Felt, K. Greenwood, and D. Wagner, "The Effectiveness of Application Permissions," *Proc. Second USENIX Conf. Web Application Development (WebApps '11),* p. 7, 2011.

[12] FriendCameo, Inc., FriendCameo, http://friendcameo.com, 2010.

[13] J. Goecks, W.K. Edwards, and E.D. Mynatt, "Challenges in Supporting End-User Privacy and Security Management with Social Navigation," *Proc. Fifth Symp. Usable Privacy and Security (SOUPS '09),* pp. 5:1-5:12, 2009.

[14] D. Goldberg, D. Nichols, B.M. Oki, and D. Terry, "Using Collaborative Filtering to Weave an Information Tapestry," *Comm. ACM,* vol. 35, no. 12, pp. 61-70, 1992.

[15] K.K. Gollu, S. Saroiu, and A. Wolman, "A Social Networking-Based Access Control Scheme for Personal Content," *Proc. 21st ACM Symp. Operating Systems Principles (SOSP '07),* 2007.

[16] R. Gross and A. Acquisti, "Information Revelation and Privacy in Online Social Networks," *Proc. ACM Workshop Privacy in the Electronic Soc. (WPES '05),* pp. 71-80, 2005.

[17] M. Hart, R. Johnson, and A. Stent, "More Content - Less Control: Access Control in the Web 2.0," *Proc. IEEE Web 2.0 Security & Privacy Workshop,* 2003.

[18] J.L. Herlocker, J.A. Konstan, A. Borchers, and J. Riedl, "An Algorithmic Framework for Performing Collaborative Filtering," *Proc. Int'l ACM SIGIR Conf. (SIGIR '99),* pp. 230-237, 1999.

[19] J.L. Herlocker, J.A. Konstan, L.G. Terveen, and J.T. Riedl, "Evaluating Collaborative Filtering Recommender Systems," *ACM Trans. Information Systems,* vol. 22, pp. 5-53, Jan. 2004.

[20] A. Herzog and N. Shahmehri, "User Help Techniques for Usable Security," *Proc. Symp. Computer Human Interaction for the Management of Information Technology (CHIMIT '07),* 2007.

[21] M. Jenkin and P. Dymond, "A Plugin-Based Privacy Scheme for World-Wide Web File Distribution," *Proc. 31st Hawaii Int'l Conf. System Sciences,* vol. 7, pp. 621-627, Jan. 1998.

[22] Y. Joshi, D. Das, and S. Saha, "Mitigating Man in the Middle Attack over Secure Sockets Layer," *Proc. IEEE Int'l Conf. Internet Multimedia Services Architecture and Applications (IMSAA),* pp. 1-5, Dec. 2009.

[23] P.G. Kelley, P. Hankes Drielsma, N. Sadeh, and L.F. Cranor, "User-Controllable Learning of Security and Privacy Policies," *AISec '08: Proc. First ACM Workshop Workshop AISec,* pp. 11-18, 2008.

[24] H.-H. Lee and W.-G. Teng, "Incorporating Multi-Criteria Ratings in Recommendation Systems," *Proc. IEEE Int'l Conf. Information Reuse and Integration (IRI '07),* pp. 273-278, 2007.

[25] M. Li, B. Dias, W. El-Deredy, and P.J.G. Lisboa, "A Probabilistic Model for Item-Based Recommender Systems," *Proc. ACM Conf. Recommender Systems (RecSys '07),* pp. 129-132, 2007.

[26] K. Liu and E. Terzi, "A Framework for Computing the Privacy Scores of Users in Online Social Networks," *Proc. IEEE Int'l Conf. Data Mining (ICDM),* W. Wang, H. Kargupta, S. Ranka, P.S. Yu, and X. Wu, ed., pp. 288-297, 2009.

[27] M.R. McLaughlin and J.L. Herlocker, "A Collaborative Filtering Algorithm and Evaluation Metric that Accurately Model the User Experience," *Proc. 27th Ann. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval (SIGIR '04),* pp. 329-336, 2004.

[28] OAuth 2.0. The OAuth 2.0 Protocol, http://tools.ietf.org/html/draft-ietf-oauth-v2-22, 2011.

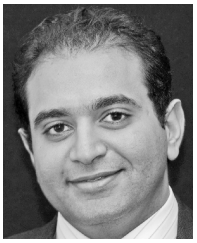
[29] N. Ramakrishnan, B. Keller, B. Mirza, A. Grama, and G. Karypis, "Privacy Risks in Recommender Systems," *IEEE Internet Computing,* vol. 5, no. 6, pp. 54-63, Nov./Dec. 2001.

[30] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, "Grouplens: An Open Architecture for Collaborative Filtering of Netnews," *CSCW '94: Proc. ACM Conf. Computer Supported Cooperative Work,* pp. 175-186, 1994.

[31] J. Riedl, "Personalization and Privacy," *IEEE Internet Computing,* vol. 5, no. 6, pp. 29-31, Nov./Dec. 2001.

[32] M. Shehab, A.C. Squicciarini, and G.-J. Ahn, "Beyond User-to-User Access Control for Online Social Networks," *Proc. 10th Int'l Conf. Information and Comm. Security (ICICS '08),* pp. 174-189, 2008.

[33] X. Su and T.M. Khoshgoftaar, "A Survey of Collaborative Filtering Techniques," *Advances in Artificial Intelligence,* vol. 2009, pp. 4:2-4:2, Jan. 2009.



Mohamed Shehab received the PhD degree in computer engineering from Purdue University. He is an assistant professor in the Department of Software and Information Systems, College of Computing and Informatics, University of North Carolina at Charlotte. He is the director of the Laboratory of Information Integration Security and Privacy. His research interests lie in network and information security, especially in the design and implementation of distributed access-control protocols to cope with the requirements of emerging distributed social networks, mobile applications, web services, and peer-to-peer environments. He is the recipient of the Google Research Award, in addition his research has been funded by US National Science Foundation (NSF), Department of Defense , and Google. He is a member of the IEEE Computer Society and the ACM.

Said Marouf received the master's degree in software engineering from the University of Wisonsin-La Crosse, in 2008. He is currently working toward the PhD degree at the University of North Carolina at Charlotte. He is a Fulbright scholar. His research interests include access control, policy optimization and management within social networks and SELinux, secure software development, and vulnerability analysis within the Java programming language execution environment.