www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 2 Issue 9 September 2013 Page No. 2737-2744

Multi-criteria Recommender systems for Open Authorization

A.Ravali, G. sudhakar
M.Tech (Computer Science) School of IT, JNTU Hyderabad, India.
ankamravali87@gmail.com

Lecturer in CSE School of IT, JNTU Hyderabad, India.
sudhakar4321@gmail.com

Abstract: **Online social networks such as Twitter, Flickr, or the Facebook have experienced exponential growth in membership in recent years. These networks offer attractive means for interaction and communication, but also raise privacy and security concerns. These online platforms 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 but fails to provide fine-grained access control. We propose an extension to the OAuth 2.0 authorization that enables the provisioning of fine-grained authorization recommendations when granting permissions to third party applications using multi-criteria recommender system. The Recommender system utilizes application based, user-based, and category-based collaborative filtering mechanisms. Our collaborative filtering (*CF*) uses the known preferences of a group of users to make recommendations or predictions of the unknown preferences for other users. 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 preferences.**

Keywords: OAuth, collaborative filtering, social networks;

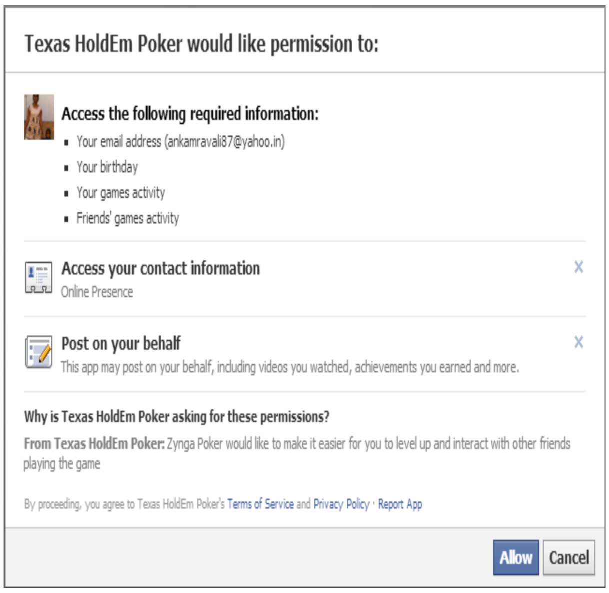
I. Introduction

Social networking sites have become rich grounds for third-party applications that utilize user online data to provide various services. 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.
In the traditional client-server authentication model, the client requests an access restricted resource (protected resource) on the server by authenticating with the server using the resource owner's credentials. In order to provide third-party applications access to restricted resources, the resource owner shares its credentials with the third-party. This creates several problems and limitations:

- Third-party applications are required to store the resource owner's credentials for future use, typically a password in clear-text.
- Servers are required to support password authentication, despite the security weaknesses inherent in passwords.
- Third-party applications gain overly broad access to the resource owner's protected resources, leaving resource owners without any ability to restrict duration or access to a limited subset of resources.
- Resource owners cannot revoke access to an individual third-party without revoking access to all third-parties, and must do so by changing their password.
- Compromise of any third-party application results in compromise of the end-user's password and all of the data protected by that password.

OAuth addresses these issues by introducing an authorization layer and separating the role of the client from that of the resource owner. In OAuth, the client requests access to resources controlled by the resource owner and hosted by the resource server, and is issued a different set of credentials than those of the resource owner.
Instead of using the resource owner's credentials to access protected resources, the client obtains an access token - a string denoting a specific scope, lifetime, and other access attributes. Access tokens are issued to third-party clients by an authorization server with the approval of the resource owner. The client uses the access token to access the protected resources hosted by the resource server.
For example, an end-user (resource owner) can grant a printing service (client) access to her protected photos stored at a photo sharing service (resource server), without sharing her

username and password with the printing service. Instead, she authenticates directly with a server trusted by the photo sharing service (authorization server), which issues the printing service delegation-specific credentials (access token).
OAuth , however does not provide necessary fine-grained access control, nor it provides any recommendation i.e., which access control decisions are most appropriate .so, in order to overcome these problems we need a recommender system which allow users to decide whether to allow individual applications to access these sensitive resources. *Time-of-use* systems prompt users to approve permissions as needed by applications at runtime, and *install-time* systems ask developers to declare their applications permission


Texas HoldEm Poker would like permission to:

Access the following required information:
- Your email address (ankamravali87@yahoo.in)
- Your birthday
- Your games activity
- Friends' games activity

Access your contact information
Online Presence

Post on your behalf
This app may post on your behalf, including videos you watched, achievements you earned and more.

Why is Texas HoldEm Poker asking for these permissions?
From Texas HoldEm Poker: Zynga Poker would like to make it easier for you to level up and interact with other friends playing the game

By proceeding, you agree to Texas HoldEm Poker's Terms of Service and Privacy Policy · Report App

a) Facebook 3rd party App Requesting Permissions.

II. PROBLEM STATEMENT

OAuth 2.0 standard provides a mechanism for third-party service providers to access end-user resources without releasing the user's access credentials to service provider. However, this does not provide fine-grained access control nor any recommendations to user before installation. An example we use throughout this paper is one of the free Facebook online games application available through Zynga Poker. The ZyngaPoker 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, and the ability to enumerate the online presence status of other users(within the first user's social network).
Once the user grants these extended permissions they cannot be realistically revoked .For example, once users provide Zynga Poker access to their e-mail addresses, they cannot realistically remove that e-mail address from Zynga Pokers'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 requirements up-front so that users can grant them during installation.
In this paper, we propose a browser Extension that implements a multi-criteria recommender-system, 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 to inform them and provide recommendations based on the collaborative decisions (grant/deny) on similar privacy requests within the user's larger social network.
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 system based on the decisions of other users within the community, and the previous decisions of an individual user.

III. PRELIMINARIES

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 thirdparty 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. 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. 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.

A. *OAuth 2.0*

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. 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 Zynga Poker application (client) access to her e-mail address on her Facebook profile without ever sharing her username and password with Zynga Poker. Instead, she authenticates the Zynga Poker application with Facebook (authorization server) which in turn provides Zynga Poker 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. 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
3) Authorization server (e.g., Facebook).

Our main focus is on steps "(A)" and "(B)" within the authorization code flow.

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.

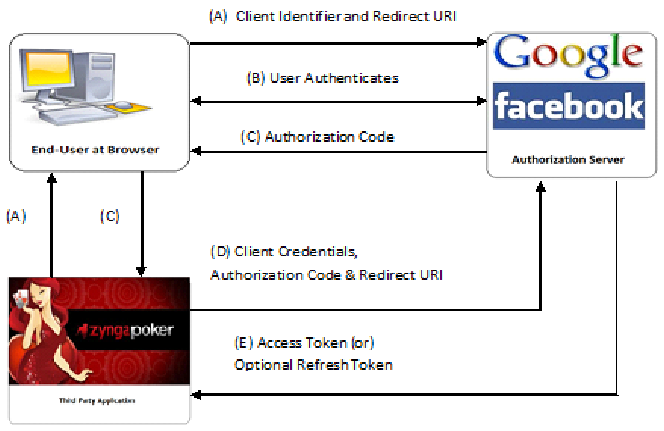

(A) Client Identifier and Redirect URI
(B) User Authenticates
(C) Authorization Code
End-User at Browser
Google
facebook
Authorization Server
(A)
(C)
(D) Client Credentials, Authorization Code & Redirect URI
zyngapoker
(E) Access Token (or) Optional Refresh Token
Third Party Application

Fig 2. Authorization code for OAuth flow.

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. 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 Zynga Poker application requests permission to post to a user's Facebook feed/wall, to access her e-mail address, and to check her friend's online/offline presence. Zynga Poker 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. Before users make their decision on the requested scope of permissions, we introduce a new level of awareness and control to the user via in-house developed browser extension..

B. Collaborative Filtering

Recommender systems are that which try to assist users in evaluating and making decisions on items by providing them opinions and prediction values as a set of recommendations. As one of the most successful approaches to building recommender systems, collaborative filtering (*CF*) uses the known preferences of a group of users to make recommendations or predictions of the unknown preferences

for other users. Collaborative Filtering is widely used and accepted as highly successful technique in recommender systems.
In this paper, as we are interested in the context of access control and user privacy, items in a collaborative filtering technique can be mapped onto individual user privacy attributes or permissions. Users can make decisions on privacy attributes, i.e., grant/deny them to third-party applications. 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.
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.

IV. EXTENDED OAUTH FLOW

We propose a new flow by extending the existing OAuth flow by adding two new modules:

1. Permission Guide: This module guides the users through the requested permissions, and shows them a set of recommendations on each of the requested permissions.

2.Recommender System: This retrieves a set of recommendations for requested permissions by following a Collaborative filtering technique.

Our extended OAuth focuses on step "(A)" of the authorization code flow in OAuth 2.0. 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.

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 permission.

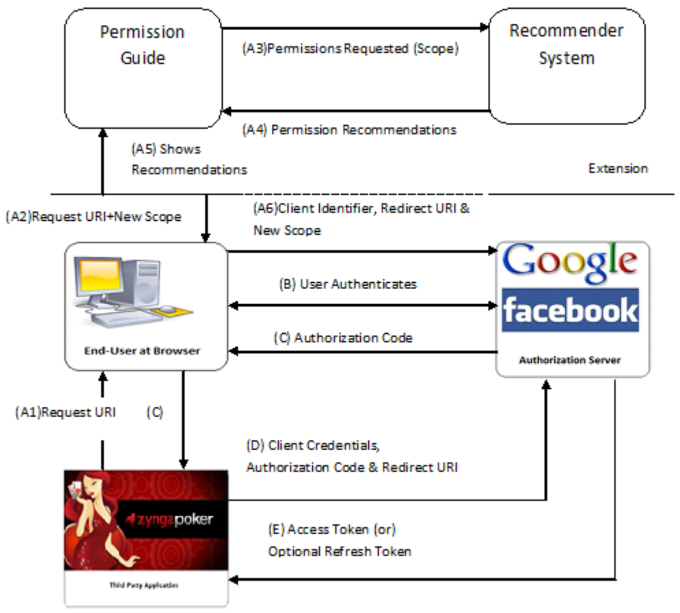

Permission Guide
(A3)Permissions Requested (Scope)
Recommender System
(A4) Permission Recommendations
(A5) Shows Recommendations
Extension
(A2)Request URI+New Scope
(A6)Client Identifier, Redirect URI & New Scope
Google
facebook
Authorization Server
(B) User Authenticates
(C) Authorization Code
End-User at Browser
(A1)Request URI
(C)
(D) Client Credentials, Authorization Code & Redirect URI
zyngapoker
Third Party Application
(E) Access Token (or) Optional Refresh Token

Fig 3., Extended OAuth Flow

A. 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.

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. An example scope´ for the Zynga Poker application could be as follows:
scope´ = post on behalf of you.
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 Recommender System 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.

This provides 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.

B. Recommender System

We propose a Recommender System component that extends upon our Permission Guide extension. Let *A*, *U*, and *P* represent the set of applications, users, and permissions, respectively. A user $u_i \in U$ can make a decision $d_i \in$ {grant, deny} on a permission $p_j \in P$ for an application $a_k \in A$. 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.

C. Collaborative Filtering

We use a multi-criteria recommender system where user recommendations are calculated per criterion. The model utilizes the set of permissions *P* as a set of criteria, i.e., each permission $p_j \in P$ represents an individual criterion within the model. The multi-criteria 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).

$$D : Users \_ Applications \rightarrow d_1 \text{ x....x } d_m \qquad (1)$$

Function(1) represents a user's overall decision on a certain application via the set of decisions made on 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 Cp_j representing user decisions on p_j for each application $a_k \in$ 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 Cp_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

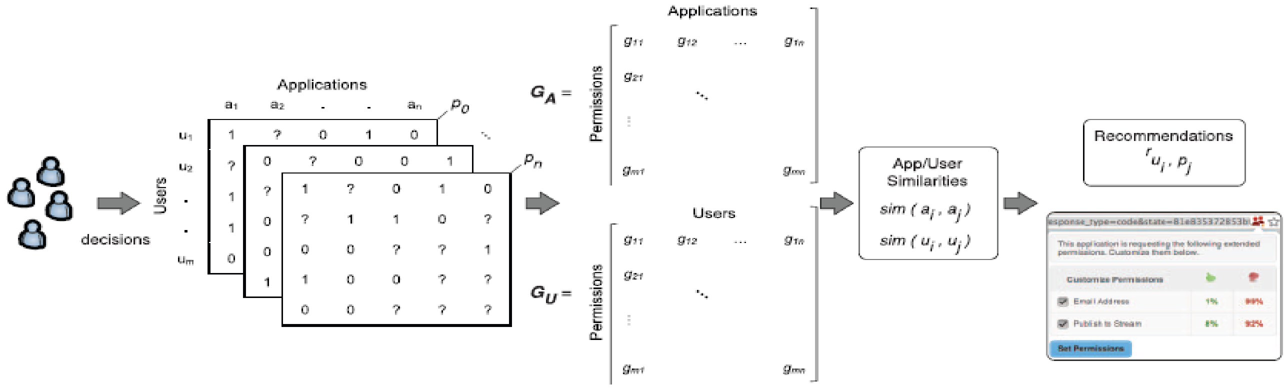


Fig 4., Collaborative Filtering Model

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 permission $\in$ set of {birthday, e – mail, location}. Our proposed model utilizes the decisions for each Cp_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 multi-criteria matrices *C* for each of the permission. By utilizing the *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 certain permission, whereas G_U captures the probability of a certain user granting certain permission.

Fig. 6 shows an example G_A matrix, with a set of applications (a1, a2, a3, a4, a5), 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 technique 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.

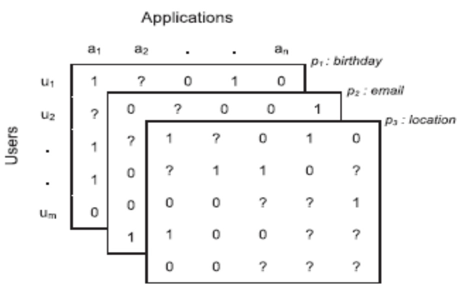
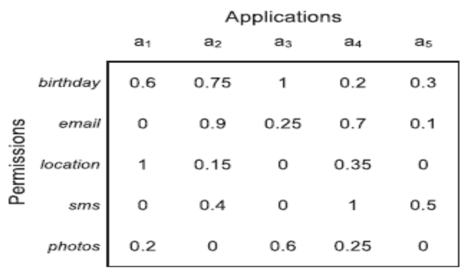


Fig 5., A three-permission (multi-criteria) model with Cp_1 , Cp_2 , *and* Cp_3 . User decisions on applications made per permission..



Permissions	Applications				
	a₁	a₂	a₃	a₄	a₅
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.

i) 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. 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 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 P.

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

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 re 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 {p_1,p_2}, 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 ai, 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.

ii) 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 (u_i) is the average probability of user u_i granting a permission in P.

$$sim(i,j) = \frac{\sum_{\forall p \in P}\left(G_U(p,i)-\overline{G_U}(u_i)\right)\left(G_A(p,j)-\overline{G_U}(u_j)\right)}{\sqrt{\sum_{\forall p \in P}\left(G_U(p,i)-\overline{G_U}(u_i)\right)^2 \sum_{\forall p \in P}\left(G_U(p,j)-\overline{G_U}(u_j)\right)^2}} \qquad (3)$$

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.

D. Prediction Model

When a user u_i, say Alice, wants to install application a_k, we calculate a set R_k, where $r_{i,j} \in R_k$ is a prediction value for permission p_j requested by a_k. $r_{i,j} \in R_k$ 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 Fig. 6 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..

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

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

In (4), reflects the average probability that permission p_j is granted over all applications in *A*, and is easily calculated via its corresponding row in the G_A matrix. Similarly, in (5), G_U (p_j) represents the average probability that permission p_j is granted over all users in 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, N represents the target application's nearest neighbors and the focus user's nearest neighbors, respectively. The size of 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 $a \in N$. In (5), $d_{j,ak}$ 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 $d_{j,ai}$ are captured via the Cp_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.

i. 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 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.

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 ai, we only consider other applications that fall into the same category as ai. Fig. 7 shows two probability matrices G and , which are extracted from the overall G_A matrix explained previously. and represent the permission probabilities for applications within the categories k and j, respectively. Let be the set of applications that belong to category *k*, and N_i be a_i*'s* nearest neighbors where. Note that a_i*'s* nearest neighbors can be found by calculating the similarities between a_i and applications within

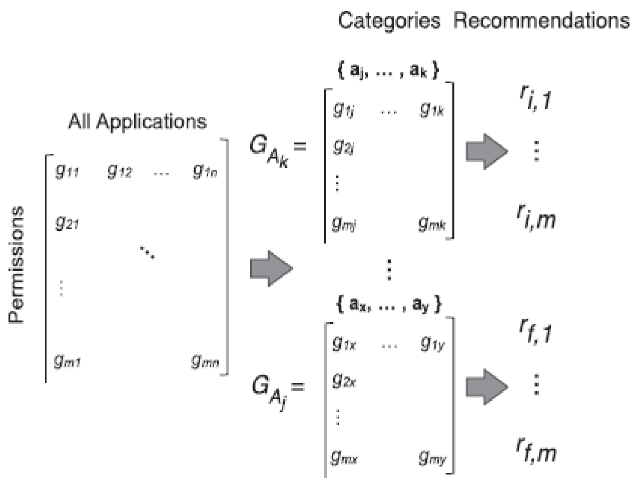


Fig. 7. Application category probability matrices. Recommendations per app category.

A_k rather than all applications in *A*. For example, in Fig. 7, the nearest neighbors for a_y are found among the set of apps {a_x . . . a_y}, and the similarities are calculated using G_{Aj}. For application a_i ϵ *A* that belongs to category *k*, we calculate recommendation predictions following:

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

where reflects the average probability that permission p_j is granted over applications in 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.

IV CONCLUSION

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 multi-criteria 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. Proposed multi-criteria recommender system leads to the preservation of irrevocable, immutable private identity attributes and the preventing of their uninformed disclosure during application installation.

REFERENCES

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

[2] 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.

[3] 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),July2010.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.