Beyond User-to-User Access Control for Online Social Networks

Mohamed Shehab[1], Anna Cinzia Squicciarini[2], and Gail-Joon Ahn[3]

[1] University of North Carolina at Charlotte, NC, USA
[2] The Pennsylvania State University, PA, USA
[3] Arizona State University, AZ, USA
mshehab@uncc.edu, acs20@psu.edu, gahn@asu.edu

Abstract. With the development of Web 2.0 technologies, online social networks are able to provide open platforms to enable the seamless sharing of profile data to enable public developers to interface and extend the social network services as applications (or APIs). At the same time, these open interfaces pose serious privacy concerns as third party applications are usually given full read access to the user profiles. Current related research has focused on mainly user-to-user interactions in social networks, and seems to ignore the third party applications. In this paper, we present an access control framework to manage the third party to user interactions. Our framework is based on enabling the user to specify the data attributes to be shared with the application and at the same time be able to specify the degree of specificity of the shared attributes. We model applications as finite state machines, and use the required user profile attributes as conditions governing the application execution. We formulate the minimal attribute generalization problem and we propose a solution that maps the problem to the shortest path problem to find the minimum set of attribute generalization required to access the application services.

1 Introduction

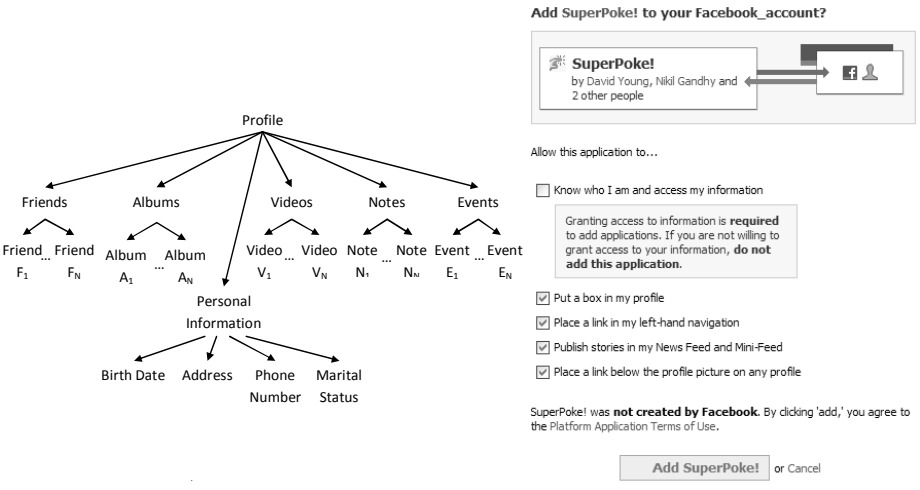
The recent growth of social network sites such as Facebook, del.icio.us and Myspace have created many interesting and challenging problems to the research communities. In social networks users self-organize into different communities, and manage their own profile, as a form of self-expression. Users profiles usually include information such as the user's name, birthdate, address, contact information, emails, education, interests, photos, music, videos, blogs and many other attributes. The structure of an example social network profile is depicted in Figure 1(a). Controlling access to the user profile information is a challenging task as it requires average internet users to act as system administrators to specify and configure access control policies for their profiles. To control interactions between users, the user's world is divided into a trusted and a non-trusted set of users, referred to as *friends* and *strangers* respectively. Furthermore, some social networks allow users to further partition the set of friends by geographical

L. Chen, M.D. Ryan, and G. Wang (Eds.): ICICS 2008, LNCS 5308, pp. 174–189, 2008.

location, social group, organization, or by how well they know them. Users are provided with group based access control mechanisms that apply access rules on the different groups of friends and strangers. Facebook [6] enables users to create a limited profile and to select which users map to that profile. For example, a user could share his wedding album with his family members and not with his colleagues from work. In addition to the issues involved with enabling fine grain access control for each user profile [5] to control data attributes viewable by other users, a yet unexplored problem is related to users' profile access from entities different from other social network users.

With the development of Web 2.0 technologies [20], online social networks are able to provide open platforms to enable the seamless sharing of profile data to enable public developers to interface and extend the social network services as applications (or APIs). For example, Facebook allows anyone to create software plug-ins that can be added to user profiles to provide services based on profile data. These features have been a great success, the most popular Facebook applications have around 24 million users as of May 2008, and competing social networking sites have moved to create their own imitation platforms. However, although these open platforms enable such advanced features, they also pose serious privacy risks. Users' profiles in fact have a great commercial value to marketing companies, competing networking sites, and identity thieves. Data mining through the development platform can potentially affect more people than screen scraping, because it exposes information that might otherwise be hidden.

Applications that are currently added to the users' profiles are given full read access to all the profiles information [6,18]. The user is able to add the



(a) Example User Profile Schema (b) App. Addition Error Message

Fig. 1. Social Networks Profiles and Applications

application only if he/she agrees to give the application access not only to his profile, but also to profile data of other users viewable through that user. In other words, the user enables the application to read information on his behalf. If the user refuses to grant full read access to the application the installation process fails. For example, Figure 1(b), shows the error message displayed by the Facebook platform when the user rejects to give the application full read access to his profile data. Basically, the application access control model adopted by the request management module is an *all-or-nothing* policy. As such, API developers have access to users' data, regardless of the actual applications' needs, leading to potentially serious privacy breaches. Such information flow is often hidden or not clear to social network users, who are often not aware of the amount of data that is actually being disclosed, since they do not really distinguish among social network users and developers outside the social network boundaries. We believe, in order to promote healthy development of social networks environments and protect fundamental individuals' privacy rights, users should be in control at any time of their data and be well informed about their usage. Applications should be given limited privileges to the user profile and only given access to the smallest set of profile data required to perform their tasks. For example, a horoscope application should be given access to only the birthday information, while a fortune cookie application that displays a random daily quote on the user's profile should not be given access to any profile data.

Although this issue has been recognized by the media [16,2,4] and by social network users, to date, no technical solution has been proposed so far. Ideally, users' should be able to take advantage of the available applications while still having a stronger control on their data. The problem is not trivial, in that it requires new access control models for APIs in social networks, as well as extending social network applications. Applications should be designed so to be customized, based on users' profile preferences and second, users should have the ability to specify the data that they are willing to reveal. Additionally, users should be able to use data privacy mechanisms such as generalization to enjoy the services provided through APIs without having to disclose identifying or private information.

In this paper we address this issue, by deploying an access control mechanism for applications in social networks. Our goal is to provide a privacy-enabled solution that is in line with social network ethics of openness, and does not hinder users' opportunities of adding useful and entertaining applications to their profiles. Our access control mechanism is based on enabling the user to specify the data attributes to be shared with the application and at the same time be able to specify the degree of specificity of the shared attributes. Enabling such a mechanism requires application to be developed to accommodate different user preferences. We model applications as finite state machines, and use the required user profile attributes as conditions governing the application execution. The challenge the user is faced with is what is the minimum set of attributes and their minimum generalization levels required to acquire specific services provided by the application. In order to address this problem we proposed the weighted

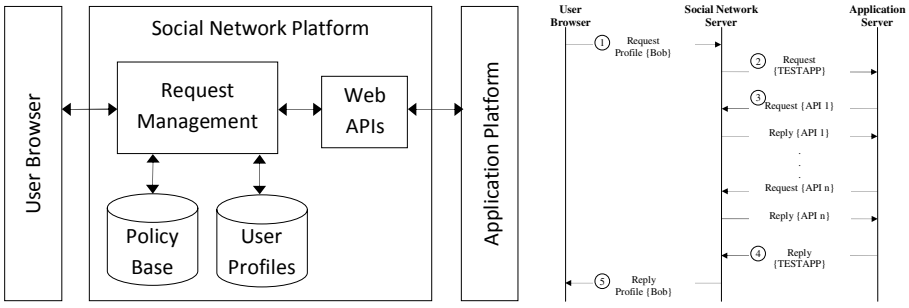
application transition system and formulated the Minimal Attribute Generalization Problem. Furthermore, we propose a solution that maps the problem to the shortest path problem to find the minimum set of attribute generalization required to access the application services.

The rest of the paper is organized as follows. In Section 2, we provide background information related to Social Network APIs. In Section 3, we introduce our developer APIs access control framework. In Section 4, we discuss how to provide customized applications. Section 5 describes the related work. The conclusion and future work are discussed in Section 6.

2 Background on Social Network APIs

With the emergence of new web technologies, and with the establishment of the Web 2.0, a large number of web sites are exposing their services by providing web programming interfaces (APIs). For example, Google Web API [12] provides a programming interface to query web pages through Google from user developed applications. Several social network web sites have released APIs that allow developers to leverage and aggregate information stored in user profiles and provide extended social network services. The exposed APIs are basically a set of web services that provide a limited and controlled view for the application to interface with the social network site. The social network application architecture includes three interacting parties namely the user, social network server, and the third party application server. Figure 2(a), shows the different blocks used in a the social networks architecture. Note that, the application server is able to connect to social network through the exported web APIs. Furthermore, these requests are filtered through the request management module which will be discussed in detail in the next section.

For example, consider an application that recommends stores in your area that are having sales. In this case, the application requires to retrieve your address, age, marital status, and gender. The address information is required to be able



(a) Social Network System Architecture (b) Application Interactions

Fig. 2. Social Network Architecture and Application Interactions

to locate shops in your region, and the other parameters are required to provide a more focused recommendation. Some other applications would not only require data from your profile but would also require data from your friends' profiles. For example, consider an application that projects your friends on an online map according to the address listed on their profiles. This application requires your address and your friend list, then for each friend it would retrieve their address.

Social networks provide mechanisms for users to customize their profiles and to add applications developed by external developers. The application provides the customized services by accessing the exported APIs. Figure 2(b), depicts the interaction stages between the user browser, social network and the third party developer application. The interaction starts when a user requests an application APP (Steps 1-2). The application server interacts with the social network server by instantiating API calls (Step 3). Upon receiving the responses of the API calls, the application server compiles and sends a response to the social network which is forwarded to the requesting user (Steps 4-5). Note that in this model the social network outsources the application development and execution to an external third party application server.

3 Developer APIs Access Control Framework

Applications require to access user's profile data to provide a service customized to the user's profile data. In this section we present our approach to enable fine grain access control [5,21] for developer's applications, to limit applications' access only to relevant user's profile data. We first provide some preliminary definitions related to applications and API set, then discuss our proposed fine grain access control framework for API based applications, and then focus on the relevant phases that characterize our approach.

3.1 Social Network Profiles and Data Sets

For the purpose of our work, the two main components of a social network are represented by applications and users' profiles.

Users' Profiles. Users' profiles are modeled as collection of data items that are uniquely associated to them. Each data item is defined over a finite domain of legal values.

Definition 1. *(User Profile) A user profile for user i, is characterized by an attribute vector $x = \{x_1, \ldots, x_n\}$, where attribute x_i takes values in a domain D^i, which also includes the null value referred to by $\perp$.*

Profile data items in our approach can be generalized to increase privacy of users. A common practice in privacy preservation mechanisms is to replace data records with suppressed or more general values [24,25] in order to ensure anonymity and prevent disclosure of sensitive data. A simple disclosure policy can simply suppress an attribute if certain disclosure criterion are met, in this case that is a

all or none policy. A generalization disclosure policy, is accomplished by assigning a disclosed value that is more general than the original attribute value. For example, the user can make the address information less specific by omitting the street and city and revealing just the zip code. Figure 3, shows an example partial value generalization hierarchy of the address attribute. We assume that domain D^i for a certain data item x_i (see Definition 1) is a partially ordered set $(D^i_j, \prec)$, where D^i_j are the attribute generalizations and $\prec$ is the ordering operator. In the domain D^i the largest element corresponds to the non-generalized attribute value and the smallest element is the most generalized value which is the suppressed value $\perp$. The domain D^i contains l_i generalization levels, an attribute generalized to the h^{th} level of generalization is denoted by D^i_h, where $0 \leq h < l_i$. Data attribute generalized to D^i_1 is more general than an attribute generalized to D^i_2, $D^i_2 \prec D^i_1$, which implies that D^i_2 discloses more information than D^i_1. Given a user profile x, by specifying generalization preferences for each

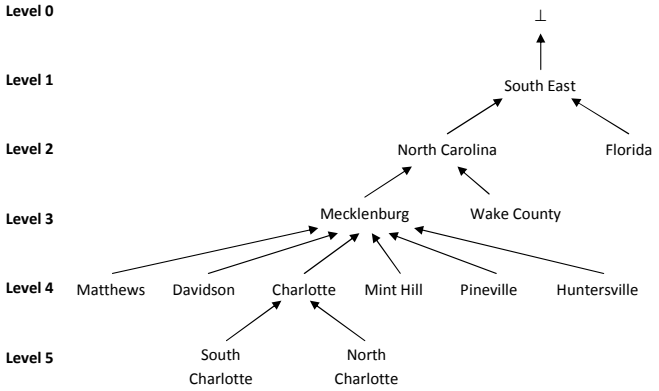


Fig. 3. A partial value generalization hierarchy of the address field

of the profile attributes the user is able to specify a different view for each application. The user generalization preferences for an application is defined by the attribute generalization vector $UP = [h_1, \ldots, h_n]$ where h_i represents the generalization level D^i_h permitted for profile attribute i. Different attributes have different disclosure sensitivity, for example some users might regard their home address more sensitive than their cell phone number. To capture attribute sensitivity, for each profile attribute $x_i \in x$ the user assigns a *sensitivity metric* Φ_i, which is specified for the non-generalized attribute $D^i_{l_i-1}$. Note that the sensitivity of an attribute x_i generalized to level h_i is proportional to $\Phi_i h_i$. Given a user generalization preference vector the $UP = [h_1, \ldots, h_n]$, the risk of attribute disclosure is proportional to $\Theta(UP) = \sum_{i=1}^{n} \Phi_i h_i$. Note that the function $\Theta()$ provides a mechanism to compare user generalization preferences. The generalization model can be applied not only to the data explicitly mentioned on the profile in addition it can be applied to the tags and the metadata that are attached to the profile data.

Applications. The building block for our model is represented by applications. Applications are composed of a set of API's which are functions called by the application.

Definition 2. *(Application API Set). Given an application App, the application API set App.apiset is the set of APIs called by application App, represented as the set* $App.apiset = \{api_1, \ldots, api_n\}$.

For example, consider a horoscope application $HoroAPP$, illustrated in Figure 4. It calls the "$user.get\_birthday()$" and "$user.get\_friends()$" APIs. The application API set for $HoroAPP$ is $HoroAPP.apiset = \{user.get\_birthday(), user.get\_friends()\}$. From the API calls the set of data set accessed by the application can be obtained by tracing the data acquired by the called API's. For example, consider an API "$user.get\_birthday()$", the profile data accessed is $\{profile.birthday\}$. Other APIs involve the processing of several profile data items for example, consider the API "$user.get\_photos\_with\_friends()$", this API returns the photos taken with friends. The API performs a join between the user friends and the user photo album meta data, in this case the data items access are $\{profile.ablums, profile.friends\}$. Accordingly, an application can be translated from a set of API calls to a set of data accesses. This set of accessed data can be then presented to the user to enable the selection of what data items to expose.

```
ExampleApplication(){
  a = get_friends(userid);
  ...
  b = get_albums(userid);
  ...
  Query = "SELECT birthday FROM user_db
             WHERE uid=userid";
  c = send_query(Query);
  ...
}
```

Fig. 4. Example of horoscope application

3.2 The Access Control Framework

Our framework adopts the *Principle of Least Privilege* [23], which requires that each principal be accorded the minimum access privileges needed to accomplish its task. In our context, principals are the application developers, and the application should be awarded access to the minimum set of profile data in order to provide the requested service. To achieve this goal we present a mechanism that enables fine grain access control on the profile data. Such a mechanism enables the application developer to select the data items required by the application and at the same time enables the user to opt-in or opt-out or generalize each of the requested data items. Specifically, our framework is characterized by three main phases: *application registration*, to register the application at the social network server; *user application addition*, to add the application in a local profile; and *application adaptation*, within which the application adapts according to the provided data items. We discuss them in what follows.

Application Registration. The application developers register the application with the social network server. The developers are required to share the application API calls and the application business state diagram describing the application process, the details of this requirement will be discussed in following sections. As part of the registration process, developers need to tag the application, by labeling each API within the application with the set of user's data items used by the application. The tags provided during this stage only refer to the user's profile data involved and do not include any external output or additional user input that may be required when executing the API. The provided application information is used to compile an *application sheet* describing the data attributes required by the application.

User Application Addition. Once the application is registered with the social network server, it becomes available for social network users to add to their social network profiles. Upon selecting the application, the application sheet is presented to the user, who is prompted with the following options for each data item required by the API: choose to opt-in, opt-out, or generalize. Intuitively, the user opts-in for the data items he is willing to disclose to the application. If the user opts-out for some data the application needs to adapt in order to be properly executed without such input. In case the generalize option is chosen for certain data item, then the user only accepts the application to employ generalized data attribute [24,25]. The user selections are input in the *user sheet*, which indicates the user access preference for the added application.

```
<APPSHEET>
 <APP id="332198764">
  <DESCRIPTION>
   <NAME> Horoscope App </NAME>
   <INFO> Provide daily horoscope
    from www.horoscope.com </INFO>
  </DESCRIPTION>
  <DATA-GROUP>
   <DATA ref="profile.birthday"/>
   <DATA ref="profile.gender"/>
   <DATA ref="profile.address"/>
  </DATA-GROUP>
 </APP>
</APPSHEET>
```

(a) Application Sheet

```
<USERSHEET>
 <APP id="332198764">
  <ALLOW>
   <DATA-GROUP>
    <DATA ref="profile.birthday.day"/>
    <DATA ref="profile.birthday.month"/>
   </DATA-GROUP>
  </ALLOW>
 </APP>
</USERSHEET>
```

(b) User Sheet

Fig. 5. Application and User Sheets

An example of XML encoding for the horoscope application is reported in Figure 5. In Figure 5(a) we report the application sheet, where birthday, gender and address are requested. In Figure 5(b) we report the user sheet in case the user opted to disclose only month and year of birth.

User Application Adaptation. At this stage the user sheet is used to generate a version of the application executable using the input obtained by the profile data items. This phase requires the application to differentiate provisioning

according to the permissible data items and their respective generalization levels. We discuss in the next section how this not trivial task is achieved.

4 Customized Application Service Provisioning

The user sheet provides a mechanism for users to specify generalization preferences on the profile attributes to restrict the data accessible to the application. On the other hand, by enabling attribute generalizations the application is faced with the problem of missing data, and might not ensure the provisioning of the request service based on the provided data generalizations. To address this issue we propose that during the application registration phase the application developer is required to provide the process execution description of the application. The process execution description describes the interactions between the composed APIs. A candidate process description language standard is BPEL (Business Process Execution Language for Web Services, also WS-BPEL, BPEL4WS) [19] which provides a rich vocabulary for expressing composition, orchestration and coordination of web services to describe the behavior of business processes. Figure 6, shows an example process execution diagram describing the service invocations and service transitions required by an application that aggregates the user's friends' addresses and projects them on Google Maps. Note that the transitions are labeled with conditions on the returned API calls. The web services composition and choreography described by BPEL can be formalized based using finite state processes (FSP) [8,22,7]. In what follows we define the application as a transition system.

Definition 3. *(Application Transition System). An application transition system is a tuple* $TS = (S, \Sigma, \delta)$*, where:*

- S *is a finite set of states. The set of states includes a single initial state* s_0 *and a finite set of final states* $F \subseteq S$*.*
- Σ *is the alphabet of operations offered by the service and the data required by this service.*
- $\delta : S \times \Sigma \rightarrow S$ *is the transition function that maps states and alphabets to another state. The transition* $\delta(s_i, \alpha) = s_j$*, represents that transition from state* s_i *to state* s_j *subject to services and data in* α*.*

The mapping function δ is used to represent the constraints required to transition from one state to another. In this paper, we focus on constraints related to the required profile data generalization levels requested by the application to enable the successful transition from a state to another. For example, an application requesting the user's address through the service *get_address*(), the application will transition to a different state depending on the generalization level of the returned address attribute. From an application perspective the user generalization preference vector specifies the permitted attribute generalization levels, which in turn dictates the set of permissible state transitions. The set of final states represents the different service levels provided by the application.

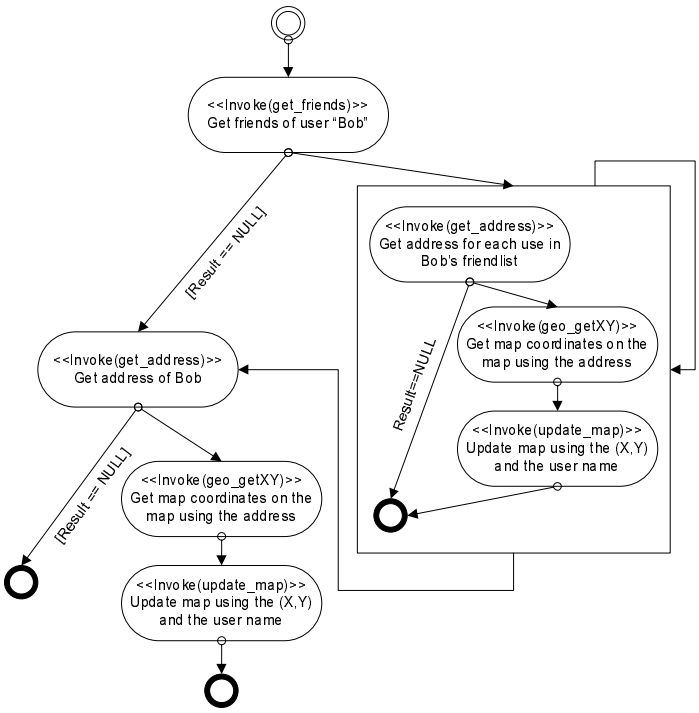


Fig. 6. Example Application Process

Definition 4. *Given an application transition system* $TS = (S, \Sigma, \delta)$ *and a user preference vector* UP*, the reduced application transition system* TS_{UP} *is defined as the tuple* $(S_R, \Sigma_R, \delta_R)$*, where:*

- $S_R = S$ *and* $\Sigma_R = \Sigma$.
- $\delta_R = \delta$ *for* $\delta(s_i, \alpha) = s_j$ *where the attributes* α *satisfies the user preference vector* UP.

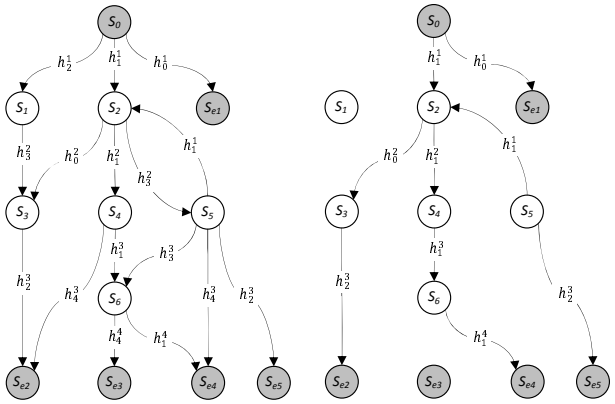
The reduced application transition system includes only the state transitions that are permitted by the user preferences. It also indicates the states that are reachable after the user preferences are applied to the application.

We model the application transition system TS as a directed graph $G = (V, E)$, where the vertices V represent the states, and the edges E represent the state transitions. The edges E are labeled with the minimum attribute generalization levels required to enable the state transition. For an edge $e \in E$ the edge label $e.h$ represents the generalization level required for the state transition. For example, in Figure 7(a) the edge (S_0, S_1) is labeled with h_2^1 indicating that the generalization level 2 is required for attribute x_1 to enable transition from state S_0 to state S_1. A user preference is said to satisfy a transition if the specified user attribute generalization level is greater than or equal to the edge generalization level. The reduced application transition system is computed by generating a

graph $G_R = (V_R, E_R)$, where $V_R = V$ and $E_R \subseteq E$ includes only the transitions E that satisfy the user preferences. Figure 7(b), shows an example reduced application transition graph for the user preference vector $up = \{h_1^1, h_1^2, h_2^3, h_1^4\}$ and the original application state diagram in Figure 7(a).

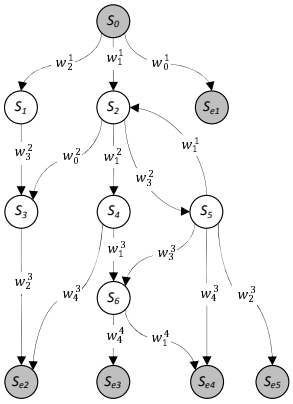
Definition 5. *(Application Service Path) Given an application transition instance TS, the path $P = \{e_0, \ldots, e_{n-1}\}$ is sequence of state transitions, where the edge e_0 starts at the initial state s_0 and the ending edge e_{n-1} terminating at a final state $s_n \in F$. The path generalization vector $g(P) = \{e_1.h, \ldots, e_{n-1}.h\}$ is defined as the set of data attribute generalization levels required to traverse this path.*

The Application Service Path represents an instance of an application execution that starts at the start state s_0 and ends at a target ending state s_n.



(a) Application State Diagram

(b) Reduced State Diagram, $up = \{h_1^1, h_1^2, h_2^3, h_1^4\}$



(c) Weighted State Diagram, $w_j^i = h_j^i * \Phi^i$

Fig. 7. Application State Diagram and User Preferences

4.1 Optimal User Application Preferences

In our framework, when trying to install an application, the user specifies an attribute generalization preferences and a target final application state. The challenge the user is faced with is to identify the minimal attribute generalization preference required to enable the application to successfully terminate to the requested final state. According to the well-known security principle of *Least Privilege* [23], which requires that each principal be accorded the minimum access privileges needed to accomplish its task, this translates to the requirement that an application should be awarded the access to the smallest set of profile attributes at the minimum generalization levels in order to provide the requested service. Formally, the minimal attribute generalization problem is defined as follows:

Definition 6. Minimal Attribute Generalization Problem, *Given an application transition instance $TS = (S, \Sigma, \delta)$, and a target final state $s_f \in F$, determine the minimal user attribute vector $UP^* = [h_1^*, \ldots, h_n^*]$ required to enable the successful transition from the start state s_0 to the final state s_f.*

The minimal user attribute vector is the vector that requires the minimum exposure of the user attributes and enables the application to transition to the target final state. Using the graph based application transition model, an application service path beginning at start state and terminating at the final target state holds the set of generalization levels required to take such a path. The minimal attribute generalization problem translates to finding the minimal application service path from the start state to the target final state in a *weighted application transition system* defined as follows:

Definition 7. *(Weighted Application Transition System). A weighted application transition system $TSW = (G, W)$ where:*

- *G is the application transition graph $G = (V, E)$, where V is the set of vertices representing the finite set of states, and E is the set of edges representing the state transitions.*
- *$W : E \times \Phi \rightarrow w \in \Re^+$ is the edge weight function that maps the edge attribute generalization labeling $E.h$ and the attribute sensitivity Φ to an edge weight w.*

Given an application service path $P = \{e_0, \ldots, e_{n-1}\}$, the path length is defined as follows:

$$\Theta(UP) = \sum_{i=0}^{n-1} W(e_i, \Phi_i) = \sum_{i=0}^{n-1} \Phi_i e_i.h$$

Given the weighted application transition system and the path length definition, the minimal attribute generalization problem simply maps to finding the shortest path from the start state s_0 to the final target state s_f. The initially

specified user preferences are used as an upper limit on the user preferences and are referred to as the upper limit user preferences $UPL = [h_0, \ldots, h_n]$. Figure 8, depicts the algorithm used to compute the minimal user attribute preferences vector. Lines 1-9, initialize the application transition graph to generate the edges that are not allowed by the specified user attribute generalization upper limits buy setting the edge weights to ∞, and the weights of the permitted transitions using the edge weight function that incorporates both the user attribute sensitivity and generalization level. Lines 10-14, initialize the distance from s_0 to other vertices, where $d[u]$ and $pi[u]$ represent the shortest distance from s_0 to u and the predecessor of u on the shortest path respectively. Lines 15-24, computes the shortest path from s_0 to all the transition states. Lines 25-34, computes the minimal user preferences vector required to transition from state s_0 to the target final state s_f.

```
Algorithm: generate_minimal_preference
Input: Application transition graph G = (V, E),
User upperlimit preferences UPL = [h_0, ..., h_n], User target state s_t
Output: User Minimal Attribute Preferences UP*

1.  V_R ← V
2.  E_R ← E
3.  //Generating the reduced graph
4.  for each e ∈ E_R
5.    for each h ∈ UPL
6.      if h ≺ e.h
7.          e.w = ∞
8.      else
9.          e.w = Φ_{e.a} * e.h
10. //Initialize distance from s_0
11. for each v ∈ V_R
12.   d[v] = ∞
13.   pi[v] = {}
14. d[s_0] = 0
15. //Computing Shortest Path from s_0
16. S = {}
17. Q ← V_R //Priority Queue on d[u]
18. while Q is not Empty
19.    u = ExtractMin(Q)
20.    S ← S ∪ {u}
21.   for each v ∈ adjacent(u)
22.     if d[v] > d[u] + w(u, v)
23.         d[v] = d[u] + w[u, v]
24.         pi[v] = u
25. //Tracing Minimal User Preferences from s_0 to s_t
26. UP* = {}
27. if d[s_t] == ∞
28.   return UP*
29. u = s_t
30. do
31.    UP* = (pi[u], u).h ∪ UP*
32.    u = pi[u]
33.  while pi[u] ≠ s_0
34. return UP*
```

Fig. 8. User Minimal Attribute Preferences Algorithm

5 Related Work

Security and privacy in Social Networks, and more generally in Web 2.0 are emerging as important and crucial research topics [15,1,14,10]. Several pilot studies conducted in the past few years have identified the need for solutions to address the problem of information leakage networks, based on interpersonal relationships and very flexible social interactions. Some social networking sites, such as FaceBook (http://www.facebook.com), have started to develop some forms of control, however the level of assurance are still limited. For example, FaceBook allows a user to join various networks (e.g., home university, home city) and control what information is released to each network. Further, a user can specify if a particular person should be "limited" from seeing particular material or blocked entirely from seeing any material. However, there is limited control over the amount of data API's can access related to user's data.

An interesting research proposal has been presented in [11], where a social-networking based access control scheme suitable for online sharing is presented. In the proposed approach authors consider identities as key pairs, and social relationship on the basis of social attestations. Access control lists are employed to define the access lists of users. A more sophisticated model has been proposed in [3]. The authors presented a rule-based access control mechanism for social networks. Such an approach is based on enforcement of complex policies expressed as constraints on the type, depth, and trust level of existing relationships. The authors also propose using certificates for granting relationships authenticity, and the client-side enforcement of access control according to a rule-based approach, where a subject requesting to access an object must demonstrate that it has the rights of doing that. However, both in both projects [11,3], the authors do not consider the issue of API's in their models, and they do not propose a method to control API's access to profile's personal data.

An ongoing research project is represented by PLOG [13]. The goal of PLOG is to facilitate access control that is automatic, expressive and convenient. The authors are interested in exploring content based access control to be applied in SN sites. We believe this is an interesting direction that we plan on investigating as extension of Private Box. Another interesting work related to ours is [9]. The authors present an integrated approach for content sharing supporting a light-weight access control mechanism. HomeViews facilitates ad hoc, peer-to-peer sharing of data between unmanaged home computers. Sharing and protection are accomplished without centralized management, global accounts, user authentication, or coordination of any kind. This contribution, although very promising does not specifically focus on SNs and thus the proposed solution, although inline with our selective approach to user's data is complementary to ours.

Some related work has also been conducted with specific focus on trust relationships in social networks. An important contribution on this topic has been proposed by [10]. The work introduces a definition of trust suitable for use in web-based social networks with a discussion of the properties that will influence its use in computation. The authors designed an approach for inferring trust relationships between individuals that are not directly connected in the network.

Specifically, they present TrustMail, a prototype email client that uses variations on these algorithms to score email messages in the user's inbox based on the user's participation and ratings in a trust network.

Our idea of transitional states was partly inspired by [17]. The authors propose a conversation-based access control model that enables service providers to retain some control on the disclosure of their access control policies while giving clients some guarantees on the termination of their interactions. Similarly to ours, the authors represent web service possible conversations as finite transition systems, in which final states in this context. Many have identified [14] the need of a new access control paradigm specific for so represent those in which the interaction with the client can be (but not necessarily) ended. We adopt a similar approach in that we represent possible applications as state machines, and we provide a labeling technique to enable the comparison of the possible application paths.

6 Conclusions

In this paper we have presented an access control framework for social networks developer applications that enables users to specify profile attribute preferences and requires applications to be designed so to be customized based on users' profile preferences. Our framework provided a privacy-enabled solution that is in line with social network ethics of openness, and does not hinder users' opportunities of adding useful and entertaining applications to their profiles. We modeled the applications as finite state machine with transition labeling indicating the generalization level required to enable application state transitions. We defined the reduced application transition system that only includes the state transitions possible with a given user generalization vector. Then we incorporated the user sensitivity metric to generate the weighted applications transition system.

Furthermore, we formalized the Minimal Attribute Generalization Problem and presented the Weighted Application Transition System which incorporates the user attribute sensitivity metric to generated a weighted graph representing the application state transitions. Using the weighted graph we transformed the Minimal Attribute Generalization Problem to the shortest path problem and provided an algorithm that generates the optimal user generalizations vector that will enable the transition to a target final state.

References

1. Acquisti, A., Gross, R.: Imagined communities: Awareness, information sharing, and privacy on the facebook. In: Privacy Enhancing Technologies, pp. 36–58 (2006)
2. CNET Blog. Exclusive: The next facebook privacy scandal (2008), http://news.cnet.com/8301-13739_3-9854409-46.html
3. Carminati, B., Ferrari, E., Perego, A.: Rule-based access control for social networks. In: OTM Workshops (2), pp. 1734–1744 (2006)
4. Wahington Chronicle. Study raises new privacy concerns about facebook (2008), http://chronicle.com/free/2008/02/1489n.htm

5. Damiani, E., Vimercati, S., Paraboschi, S., Samarati, P.: A fine-grained access control system for XML documents. ACM Transactions on Information and System Security 5(2), 169–202 (2002)
6. Facebook (2007), http://www.facebook.com
7. Foster, H., Uchitel, S., Magee, J., Kramer, J.: Ltsa-ws: A tool for model-based verification of web service compositions and choreography, pp. 771–774 (May 2006)
8. Foster, H., Uchitel, S., Magee, J., Kramer, J., Hu, M.: Using a rigorous approach for engineering web service compositions: a case study, vol. 1, pp. 217–224 (July 2005)
9. Geambasu, R., Balazinska, M., Gribble, S.D., Levy, H.M.: Homeviews: peer-to-peer middleware for personal data sharing applications. In: SIGMOD Conference, pp. 235–246 (2007)
10. Golbeck, J., Hendler, J.A.: Inferring binary trust relationships in web-based social networks. ACM Trans. Internet Techn. 6(4), 497–529 (2006)
11. Gollu, K.K., Saroiu, S., Wolman, A.: A social networking-based access control scheme for personal content. In: Proc. 21st ACM Symposium on Operating Systems Principles (SOSP 2007) (2007); Work in progress
12. Google Code. Google's Developer Network, http://code.google.com/
13. Hart, M., Johnson, R., Stent, A.: More content - less control: Access control in the Web 2.0. Web 2.0 Security & Privacy (2003)
14. Hogben, G.: Security issues and recommendations for online social networks. ENISA Position Paper N.1 (2007)
15. IEEE. W2SP 2008: Web 2.0 Security and Privacy (2008)
16. Irvine, M.: Social networking applications can pose security risks. Associated Press (April 2008)
17. Mecella, M., Ouzzani, M., Paci, F., Bertino, E.: Access control enforcement for conversation-based web services. In: WWW Conference, pp. 257–266 (2006)
18. MySpace (2007), http://www.myspace.com
19. OASIS. OASIS WSBPEL TC Webpage, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel
20. O'Reilly, T.: What Is Web 2.0. O'Reilly Network, pp. 169–202 (September 2005)
21. Rizvi, S., Mendelzon, A., Sudarshan, S., Roy, P.: Extending query rewriting techniques for fine-grained access control. In: SIGMOD 2004: Proceedings of the 2004 ACM SIGMOD international conference on Management of data, pp. 551–562. ACM, New York (2004)
22. Salaun, G., Bordeaux, L., Schaerf, M.: Describing and reasoning on web services using process algebra, pp. 43–51 (June 2005)
23. Saltzer, J., Schroeder, M.: The Protection of Information in Computer Systems. Proceedings of the IEEE 63(9), 1278–1308 (1975)
24. Samarati, P., Sweeney, L.: Generalizing data to provide anonymity when disclosing information (abstract). In: PODS '98: Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, p. 188. ACM, New York (1998)
25. Sweeney, L.: k-anonymity: a model for protecting privacy. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 10(5), 557–570 (2002)