Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für
Technische Informatik und
Kommunikationsnetze

Master Thesis

Privacy in Online Social Networks

Christoph Renner
<chrigi.renner@gmail.com>

Advisors

Dr. Thomas Duebendorfer, Google Switzerland GmbH

Martin Burkhart, ETH Zurich

Supervisor

Prof. Dr. Bernhard Plattner, ETH Zurich

Communication Systems Group (CSG)
Computer Engineering and Networks Laboratory (TIK)
Department of Information Technology and Electrical Engineering (D-ITET)
Swiss Federal Institute of Technology (ETH Zurich)

&

Google Switzerland GmbH

12.10.2009 - 12.04.2010

Acknowledgments

I would like to thank:

- Prof. Dr. Bernhard Plattner from ETH Zurich for supervising this thesis.
- Martin Burkhart for being my advisor from ETH Zurich.
- Dr. Thomas Duebendorfer for hosting me at Google and for supporting me with valuable input, feedback and many fruitful discussions.
- Mark Weitzel and Tyrone W. Grandison from IBM for their feedback on the proposals to extend OpenSocial.
- Sachin Shenoy and Shishir Birmiwal from Orkut for helping me work with Orkut.
- Lane LiaBraaten and Rahul Kulkarni from Google for their feedback on the OpenSocial proposal.
- Jason Cooper from Google for integrating my change lists into the OpenSocial Java Client.
- Google Switzerland GmbH and especially the Milliways team for providing a productive work environment and excellent food.

Abstract

Online social networks penetrate today's life more and more. Much appeal comes form the personal data which is shared using those networks. This potentially sensitive data can expose the users to threats such as embarrassment or identity theft. This thesis, after looking at the current state of research in the field of privacy in online social networks, proposes modifications to the OpenSocial specification to add the first privacy features to this cross-platform API for applications interacting with online social networks. Our proposals define access control lists and their use for albums, media items and activities. These changes provide application developers the necessary means to give the user more control over his data in online social networks and more transparency of who can access that data. To demonstrate the usefulness of the proposed changes, an Android application was created and the proposed API was implemented as a part in Google's social network platform Orkut.

Zusammenfassung

Soziale Netzwerke im Internet nehmen in der heutigen Zeit eine immer wichtigere Rolle ein. Ein wichtiger Bestandteil dieser Netzwerke sind die persönlichen Informationen, welche darin ausgetauscht werden. Diese möglicherweise heiklen Daten können die Benutzer Risiken wie zum Beispiel Blossstellung oder Identitätsdiebstahl aussetzen. Diese Arbeit begutachtet zuerst den aktuellen Stand der Forschung im Gebiet von Privatsphäre in sozialen Netzwerken und schlägt danach Erweiterungen zum OpenSocial Standard vor. Diese Erweiterungen sind die ersten Datenschutzfunktionen in diesem Plattform übergreifenden Standard für Anwendungen für soziale Netzwerke. Die Erweiterungen definieren Zugriffskontrolllisten für Albums, Media Items und Activities, welche es Entwicklern ermöglichen dem Benutzer mehr Kontrolle über seine Daten zu geben und ihm transparenter mitzuteilen wer auf seine Daten zugreifen kann. Um die Nützlichkeit dieser Erweiterungen aufzuzeigen wurde eine Android Anwendungen geschrieben. Des weiteren wurden die vorgeschlagenen Erweiterungen in Googles sozialem Netzwerk Orkut implementiert.

Contents

1 Introduction

1.1 Online Social Networks

In the past few years online social networks have become highly popular. As of February 2010, Facebook had more than 400 million active users. Every day 50% of those active users logged on to Facebook [9]. In most online social networks users create profiles which often contain details about their personal lives. These profiles are shared with friends, networks and sometimes also with strangers. Some online social networks also provide a platform to share multimedia content like photos and videos. Facebook for instance is one of the largest Photo Sharing Sites worldwide. Every month 3 billion photos are being uploaded to Facebook [9].

1.2 Risks in Online Social Networks

The personal information shared in online social networks can harm the user in often unexpected ways. For example, in the United States of America, knowing someone's birth date and state of birth is often enough to predict that person's Social Security Number. This might allow for identity theft because the Social Security Number is often used for identification [3]. Photos uploaded to online social networks can also be harmful for someone when they fall into the wrong hands. Uploading photos of a wild party might be harmless when shared with friends who were also at that party but it might not benefit the applicant if those photos fall into the hands of his recruiter.

A Google search for "lost job because of Facebook" shows that the threats of using online social networks are real and nothing rare. Especially ranting about one's boss or job seems to be a pretty common reason for getting fired. A case which shows that sharing photos can have harmful consequences is the story of Nathalie Blanchard who has been on sick leave because of depression. She lost her insurance benefits because, according to the insurance company, the photos on her Facebook profile prove that she is no longer depressed [16].

1.3 Problem Statement

There's a lot of confusion about what is handled as public, semi-public or private information in online social networks. While several social networking sites offer data sharing controls, there's no standard way of checking and controlling which personal information is shared with whom.

1.4 Outline

While much of the privacy research in online social networks (see Section 2) is rather theoretical, I wanted my thesis to have a practical impact which helps users to share information in online social networks in a comfortable and secure way. To have this impact OpenSocial seems to be a good platform to work with since it's developed by an open community where anyone can contribute. Changes to OpenSocial also have an impact on a many online social network users due to the large number of providers which support OpenSocial.

This thesis proposes an extension to the OpenSocial API specification to enable for provider independent applications which give the user more transparency about who gets access to his data. In addition to increased transparency the proposal also gives the user more control over his data by offering a provider independent way to modify data sharing settings.

This Master's thesis will look at the current state of privacy controls in popular social networks and propose new ways for controlling the personal privacy risk. Following a privacy control analysis in popular social networks, a proposal to extend OpenSocial with the first privacy features is presented. As a reference implementation those features were implemented in the OpenSocial endpoint of Google's social network Orkut. In order to provide a use case to the proposed changes Photocial, a photo sharing application for Android, was implemented.

2 Literature Review

With the online social networks gaining importance over the last few years, many research papers have been written on privacy in online social networks. This section provides an overview of the more relevant papers.

The social graph in online social networks is of significant interest to researchers in various application domains such as marketing or psychology. Of course such social graphs also contain privacy sensitive data and therefore these graphs cannot be published in their raw form. Backstrom et al. show that anonymizing graphs by simply removing identifiers does not guarantee privacy [4]. In [12], Kun Liu et al. propose an algorithm which anonymizes a social graph to guarantee a certain degree of privacy while still keeping the graph's properties, which are of interest when analysing the graph.

Another research area is how to provide the user with controls to protect his privacy by specifying which data should be accessible by whom. Maximilien et al. propose an algorithm to compute a user's privacy risk similar to the credit score which is used to describe a person's creditworthiness. This algorithm also allows to change the visibility of information to adjust a user's privacy risk to a certain value [14]. Bonneau et al. [5] on the other hand describe how online social networks promote their privacy policies and settings with a model in which the provider needs to satisfy the privacy aware users by offering sufficient controls for privacy. Because even if the privacy aware users are a minority of the user base they still influence the other users with blog posts or news articles. Bonneau also shows that providing more privacy controls and making more assurances about the user's privacy being preserved can make less privacy aware users less comfortable about using the service. Kelley et al. take another approach by introducing a "Nutrition Label" for privacy. Similar to the nutrition facts label this privacy label shows the user how an Internet site treats the user's data. In contrast to the privacy policies used today such as P3P [20] such a label could be more easily understood by uneducated users.

In today's web based applications like online social networks, the users need to ultimately trust the service provider not to misuse the users' information. Once the user sends the data to the provider's server the user cannot control in which way the provider will actually use the data. Leucio Antonio Cutillo et al. proposed a decentralized social networking service [7]. In contrast to other peer to peer networks they use the social graph to ensure collaboration by assuming that users with a "friend" relationship are willing to help each other.

Almost all methods to protect the user's privacy only work under the assumption that an attacker is limited to certain types of attacks. Wondracek et al. show in their

tech report [21] what can be achieved when one finds a new method to undermine the user's privacy. They use a well-known technique of web browser history stealing [11] to determine the group membership of a user. Using this group membership information they have a good chance to de-anonymize the user. Because the used history stealing attack can be performed by any website the user visits, any of those websites can possibly de-anonymize the user.

2.1 Privacy Controls in APIs

When Facebook launched its Facebook Platform in 2007 this dramatically changed Facebook's nature. It transformed from a website which was maintained by Facebook to a platform on which companies and software developers can write applications which use the social structures to provide services to Facebook members. Soon after this launch Google announced *OpenSocial*, a similar application programming interface (API) which is described in Section 3. In addition to the initial APIs which allowed for applications to run within the online social network provider's website, APIs for directly communicating with the social network were added to enable for applications to run on other servers to connect with social networks. In addition to web applications also desktop applications and applications running on mobile phones can use those server to server APIs which allow for many interesting applications. This section gives an overview about the privacy controls available to third party developers using those APIs.

2.1.1 Facebook RESTful API

Facebook provides a proprietary RESTful API, which allows developers to interact with the Facebook platform. For some methods, it supports all of the privacy controls provided by Facebook's web interface.

One method that supports the full set of privacy options also provided by the web interface is the `photos.createAlbum` method, which is for creating new albums. It has two parameters to control the visibility of the new album: `visible` and `privacy`. The `visible` parameter can be one of `friends`, `friends-of-friends`, `networks` or `everyone`. The `visible` parameter is only supported for legacy reasons, new applications should use the more powerful `privacy` parameter. The `privacy` parameter consists of key/value pairs which are described in Table 1. This parameter is more complex than the `visible` parameter but is able to represent all the privacy settings which can be specified using the web UI.

Unfortunately, the `photos.getAlbums` method which retrieves the user's albums does

Key	Value
`value`	One of `EVERYONE`, `CUSTOM`, `ALL_FRIENDS`, `NETWORKS_FRIENDS`, `FRIENDS_OF_FRIENDS` or `SELF`.
`friends`	For `CUSTOM` settings, specifies which users can see the album. Can be one of `EVERYONE`, `NETWORKS_FRIENDS` (when the album can be seen by networks and friends), `FRIENDS_OF_FRIENDS`, `ALL_FRIENDS`, `SOME_FRIENDS`, `SELF`, or `NO_FRIENDS` (when the object can be seen by a network only).
`networks`	For `CUSTOM` settings, a comma-separated list of network IDs that can see the album.
`allow`	When friends is set to `SOME_FRIENDS`, a comma-separated list of user IDs and friend list IDs which can see the album.
`deny`	When friends is set to `SOME_FRIENDS`, a comma-separated list of user IDs and friend list IDs which cannot see the album.

Table 1: The `privacy` parameter to create albums using the Facebook RESTful API

not return a field which is able to represent the more complex privacy settings. Instead it only returns a `visible` field which can be one of `friends`, `friends-of-friends`, `networks`, `everyone` or `custom`. If `visible` equals `custom` no further information is provided. This is especially annoying because an application cannot distinguish between an album being shared with a lot of people in networks and an album not shared at all because both cases are mapped to `custom`. This is even true for albums which were created by the application itself because one cannot be sure that the user has not changed the setting using another application or the web UI.

Some other methods are also missing the appropriate parameters to support the same privacy functionality as is provided by the web UI. Status updated as an example can individually be controlled by privacy settings but the methods `status.set` and `users.setStatus` have no arguments to specify who should be able to see the status update.

2.1.2 OpenSocial

In contrast to the previously discussed APIs, OpenSocial is a set of APIs which is not being developed by a single online social network. See Chapter 3 for a high level overview over OpenSocial.

Unfortunately OpenSocial was not designed with privacy in mind. It provides no way to access privacy settings. It does not provide any information about who can access

which resource and also does not allow to specify with whom a resource is shared. When creating a new resource it also does not allow to differ from the default privacy setting, which is in general not known and not specified in OpenSocial.

However, MySpace provides access to privacy settings for albums through their OpenSocial implementation in a proprietary extension [15]. The API allows to retrieve and set the privacy level for albums which specifies who can view the photos inside that album. Possible values for the privacy level are `me`, `friendsonly` and `everyone`. `me` allows only the person who uploaded the photo to view it, `friendsonly` allows all the friends of the user who uploaded the photo to view it and `everyone` lets everyone in the Internet view the photo. Unfortunately, this proprietary extension is only supported by MySpace and not part of the OpenSocial standard and therefore cannot be used across different platforms.

2.2 Privacy Metrics

Measuring privacy in social networks is a difficult task. It's not inherently clear which information can lead to considerable damage such as identity theft. Other risks are even harder to assess: comments and pictures which are harmless for some people can be harmful for others. One possible example is criticism against a government or religion. In some cultures and countries such criticism is widely accepted whereas in other places someone can get in severe troubles for doing so.

2.2.1 Two simple metrics

One common approach to define risk is by the following formula:

$$\texttt{risk} = \texttt{negative consequence} \times \texttt{likelihood} \tag{1}$$

the problem with using this definition in the context of online social networks is that often `consequence` and `likelihood` are both unknown. For example how can one define the consequence of leaking an embarrassing picture? The consequences could range from just feeling embarrassed in front of friends to ruining one's career if one occupies an important position in economics or politics. An often mentioned danger of using online social networks is that when posting information about vacations abroad, burglars could use this information to learn when a house is uninhabited to decide when to rob the house [2]. On the internet countless news reports and blog posts can be found stating that this is possible but no statistics about how often this actually happens have been found. Not even a single case for which this method has been used was found.

An even simpler metric to measure privacy is the *number of people* which can access the information at a given time. Of course this metric can only ensure a certain privacy level when the number of people with access is small enough such that the user which shares the information knows and trusts all of them. However this metric is often used in the real world. Most people discuss private information in a train because they know that only the people in the same compartment will hear them talking whereas they would never discuss the same things when talking into a microphone such that everyone in the train can listen. To interpret this metric someone, at best the sharing person itself, must assess the risk of sharing the information with a group of people of that size. In addition to the size, the composition of the group needs also to be taken into consideration but the simple metric of the group size still conveys a feeling for how "public" or "private" some piece of information is.

2.2.2 Privacy Risk Score

Maximilien et al. from the IBM Almaden Research Center define the privacy risk for a user based on sensitivity and visibility of profile items which are derived from sharing settings of the users [14]. The model used for this computation is a social network in which each of the N users has n profile items. For each item i the user can choose whether he wants to share that item. These sharing settings can be represented as a binary $n \times N$ matrix R where $R(i, j)$ is the entry in the i-th row and j-th column. The profile item i of the user j is shared if $R(i, j) = 1$.

They define the privacy risk score based on the following two premises:

1. the more sensitive data a user reveals, the higher his privacy risk is
2. the more people know some piece of information about the user, the higher his privacy risk is

From these two premises follows their definition of the privacy risk as a monotonically increasing function of two parameters: the sensitivity and visibility of the user's profile items. The sensitivity of profile item i is denoted as β_i and P_{ij} stands for the visibility of that item i of the user j. The privacy risk $PR(j)$ for the user j is then expressed as the sum of the individual privacy risks of each profile item of that user. To compute the risk of a single profile item the product of its visibility with its sensitivity is used. This leads to the following equation:

$$PR(j) = \sum_i \beta_i P_{ij} \tag{2}$$

The sensitivity β_i is computed as the proportion of users which are reluctant to share the profile item i:

$$\beta_i = \frac{N - |R_i|}{N} \tag{3}$$

where $|R_i|$ is the number of users which share item i. This is exactly the numbers of users j who have set $R(i, j) = 1$. The visibility P_{ij} is an estimate of the probability that $R(i, j) = 1$. The simplest way to express the visibility of an item is to set $P_{ij} = R(i, j)$ which is the observed visibility. The drawback of this simple solution is that the sharing settings of all users must be known to compute the privacy risk score of a single user. In [13] Kun Liu et al. extend this metric by using a probabilistic model to compute the privacy risk score based on the sharing settings of a subset of the users.

An interesting feature of this metric introduced by Maximilien et al. [14] is that it enables the propagation of privacy settings among users. One can compare a user's privacy risk to the aggregated privacy risk of the user's social graph. If the user's privacy risk is too high, possible privacy settings can be proposed to the user to match the desired privacy risk score. They wrote the Privacy-aware MarketPlace (PaMP) Facebook application [6] as a proof of concept implementation using Equation 2 as a definition for a privacy risk score.

2.3 Understanding the Sharing Model

To have control over his personal data in the online social network the user needs, in addition to using the privacy controls provided by the online social network, also to understand how the underlaying sharing model works. Due to lack of documentation one can often only understand the sharing model by performing experiments. For example: I wondered whether the action of tagging someone in a photo in Facebook transfers some rights to the user who is tagged. This section describes what I did to answer this question and what I found out.

At first the Facebook Help Centre [8] was searched for an answer. The only relevant information found was the answer to the question: "I am able to view a non-friend's entire photo album by clicking through from a photo that my friend is tagged in. Is this violating their privacy settings?" and its answer: "It's important to remember that the 'tagging' feature does not allow users to see photos that they wouldn't normally be able to see. . . . ". From this statement one could expect that someone cannot access a private photo of someone else in which he is tagged. To see whether this expectation matches Facebook's actual behavior I conducted a small experiment with two accounts in a friend relationship. They need to be friends such they can tag each other in their photos. Using one account I created a new album with the privacy setting "Only me"

which makes the album and the photos in that album only visible to the account which creates the album and uploads the photo. Than I tagged the user which has no access to this album in a photo which I uploaded into that album. The user which has been tagged then got a notification that he has been tagged with a link to the photo which is now accessible to him because of the "tagging" feature. This behavior seems to conflict with the statement made in the Help Centre.

Although some notification that the user who is being tagged gets access to the private photo would be a nice feature it still makes sense to grant the user access to the photo since the owner explicitly tagged him. However a second experiment showed that in addition to being able to view the image he also gets the right to tag new persons in that photo. Those people then also get access to the photo and could possibly tag further people to pass on the right to access the photo without giving the owner a chance to prevent that. All he sees are notifications that some people have been tagged in his photo.

3 OpenSocial

OpenSocial [19] is a set of common APIs to write applications which interact with an online social network initially released by Google in 2007. When OpenSocial was announced publicly already 18 sites which have about 200 million users in total have committed to support OpenSocial [10]. As of November 2008, one year after the launch, OpenSocial was supported by 20 sites with 600 million users [1]. Since it's launch, the API specification was developed by a community which treats it like an open source software project. The four basic principles are:

- Participation is open to anyone
- Decisions are made on the spec list (not behind closed doors)
- All proceedings are captured in a public archive
- Individuals represent themselves, not companies

The OpenSocial 1.0 specification [18] is divided in three parts: Data Specification, API Server Specification and Gadget Specification. The following three sections provide a short overview over the Data, Gadget and API Server Specification.

3.1 OpenSocial Data Specification

The OpenSocial Data Specification defines some basic data types such as `Boolean`, `Domain-Name`, `Object-Id` and more complex data types which are defined as objects which contain fields which map from a name to a value. Values used in such objects can have any type defined by OpenSocial. This includes other objects, arrays of any type and basic types such as booleans and strings.

To give an example of such a data object, the `Album` type is used. Table 2 shows the structure of an album as it is defined in OpenSocial 1.0. The album object bundles information about the album like owner, title, description, thumbnail and the location which should be associated with this album and some statistic information about the contents of the album like the number of media items, their mime-types or media types. Possible media types are image, video or audio. The id field is used to specify the album in methods which operate on an album.

Field Name	Field Type	Description
description	string	Description of the album
id	Object-Id	Unique identifier for the album.
location	Address	Location corresponding to the album.
mediaItemCount	integer	Number of items in the album.
mediaMimeType	Array <string>	Array of strings identifying the mime-types of media items in the Album.
mediaType	Array <string>	Array of MediaItem types, types are one of: audio, image, video.
ownerId	Object-Id	ID of the owner of the album.
thumbnailUrl	string	URL to a thumbnail cover of the album.
title	string	The title of the album.

Table 2: The OpenSocial Album Type as defined in OpenSocial 1.0

3.2 OpenSocial Gadget Specification

Gadgets are software components which can be embedded into various contexts such as standalone web pages, web applications or even other gadgets. The context in which they are embedded is denoted "container". They're written in HTML, CSS, JavaScript and XML.

The Gadget XML specifies which features a Gadget requires. This can be a specific OpenSocial version or OpenSocial features which are optional for the container to support. The Gadget XML usually also contains links to HTML and CSS files which usually define the UI of the Gadget and JavaScript code which provides the program logic behind the Gadget. The OpenSocial Gadget specification also provides a JavaScript API to access data which is stored in the container. All the JavaScript API parts which are relevant for this thesis are a direct mapping to the respective API Server parts which are described in the next section.

3.3 OpenSocial API Server Specification

The OpenSocial API Server specification defines two protocols to interact with OpenSocial container servers outside of gadgets on a web page: A RESTful protocol which must be supported by all servers compatible to the specification and the RPC protocol which is optional. Both protocols basically support the same operations and data representations. All types can be represented either in JSON, XML or Atom / AtomPub. The

Service	Description
Cache	Used to manage the resources cached by the container.
System	Retrieve information about supported services and operations.
People	Gives access to data about users of the container. E.g. retrieving a user's friend list or updating profile information.
Groups	Groups are sets of users of the container.
Activities	Activities are short summaries or notifications of timestamped events.
AppData	AppData provides a data store that applications can use to read and write user-specific data.
Albums	An Album is a collection of media items.
MediaItems	A MediaItem represents an image, movie or audio file.
Messages	The Messages service allows to send messages to users.

Table 3: OpenSocial API Server Services

specification describes a generic set of mapping rules to the data from one format to another. I'll primarily use the RPC protocol and the JSON data representation in this documentation.

The API functions are divided into services. Each service provides access to a certain type of data within the container. See Table 3 for a list of available services. The majority of the methods provided by those services are data oriented. The four most important methods provided by almost all services are `get`, `update`, `create` and `delete`. The `get` method retrieves one or several resources. When retrieving a collection, e.g. all friends of a user, `get` has built in support for pagination, that is it allows for fetching only a subset of the collection specified by start index and number of elements. Every resource returned by the `get` method has a unique ID which can be used to delete the resource via the `delete` method and modify it using the `update` method. Some `create` operations are just as simple as calling `create` with the resource to create as argument but when additional data, which is not part of the actual resource, needs to be uploaded it is a bit more complex than that. One such case is creating a new media item for an image. The image data itself is not part of the resource which contains only text information. Instead the resource contains a URL which can be used to fetch the image. To upload an image one has three available methods: Two Step Upload, Shortcut Upload and Multi-Part Upload. To be standard compliant, the server needs to implement the Two Step Upload. The two other methods are optional. In the Two Step Upload method at first the image data is sent to the server using the HTTP POST method. The server then returns a skeleton media item which can be used to add additional data using the `update` method. When using the Shortcut Upload Method one also uses the POST method to

upload the image but passes all the additional media item data in the query string of that POST request. These two methods are easy to map to the REST API but could also be implemented by an RPC endpoint. The Multi-Part Upload on the other hand was designed to be used exclusively in the RPC API. The body of the HTTP request consists of a MIME Multipart message. One part consists of the normal RPC payload, which can contain links to other parts of the Multipart message. To upload a photo one needs to create a multipart message where one part is the method `mediaItems.create` and the other part contains the image data.

4 Design of OpenSocial Privacy API

During this thesis three proposals to modify the OpenSocial specification have been sent to the OpenSocial and Gadgets Specification Discussion forum [17]. This chapter describes the design of these proposals which are attached to this report as appendices.

While designing the OpenSocial Privacy API the main objective was to create a simple API which is nevertheless powerful enough to support the sharing settings supported by today's online social networks. In addition to supporting today's sharing settings, possible future sharing settings were also kept in mind. The API should be simple to use for developers who write applications using OpenSocial such that they can implement features using the privacy API without much effort. If the API is too complicated the majority of developers probably wouldn't use it. In addition to being easy to use on the client side the API should also be easy implementable for OpenSocial containers to get the support of the OpenSocial community to increase the chances that the API will be included in the next version of the OpenSocial specification. Once the API is part of the specification, online social networks are also more likely to implement the privacy API if it's possible without much effort.

In addition to being simple, making the proposal generic enough to be reused in different parts provided by OpenSocial was another main goal. Therefore the proposal has been split in three parts: one describing a generic ACL structure for OpenSocial and two providing access controls to media items and activities using this generic ACL. A fourth proposal which uses ACLs to control access to profile information was declared as further work. However the last section of this chapter gives some thoughts on how to define such an API.

The proposal to extend the OpenSocial specification was first based on version 0.9 of the OpenSocial specification. During this thesis the OpenSocial community released version 1.0 of the specification, which in contrast to version 0.9 is split into several parts which are described in Section 3. As a consequence of the release of version 1.0, the proposal had to be rewritten to match the new structure and use the augmented BNF syntax, also introduced in OpenSocial 1.0, to describe the data structures.

4.1 Privacy Levels in Online Social Networks

In order to create an API which supports the privacy settings supported by the online social network sites in their web UIs, the UIs of a few important online social network sites have been examined. The survey conducted during this thesis, does also include online social network sites which do not support OpenSocial to get a broader overview

	Orkut	MySpace	Hi5	Netlog	Flickr	StudiVZ	Facebook
Everybody	✗	✓	✗	✓	✓	✗	✗
All Users	✓	✗	✓	✗	✗	✓	✓
Friends of Friends	✗	✗	✗	✗	✗	✗	✓
Friends Only	✓	✓	✓	✓	✓	✓	✓
Family	✗	✗	✗	✗	✓	✗	✗
Me Only	✗	✓	✗	✓	✓	✓	✓
Selected Friends	✓	✗	✗	✗	✗	✗	✓
Email	✓	✗	✗	✗	✗	✗	✗
Others	✗	✗	✗	✗	✗	✗	✓

Table 4: Privacy levels for photo albums

about the possible sharing settings.

Table 4 shows the available sharing options of some online social networks with support for photo sharing. “Everybody” stands for everybody with access to the Internet whereas “All Users” stands for everybody who has a profile on the online social network. The difference between these two sharing levels is often negligible since everybody can create a profile without much effort. However search engines cannot index albums which require a profile to be visible and those albums therefore do not show up in search results. “Friends Only” includes those users which are in a friendship relation within the social graph of the person who creates the album. “Friends of Friends” means all the friends and their friends. In the social graph of the owner that are all the users who have a distance of at most two to the owner. “Family” is the set of users which are in a family relationship with the owner. Flickr does not require that “Family” is a subset of “Friends”. Any contact can be flagged as being part of the “Family” group. “Me Only” allows no one else than the owner to see the album and its contents. Using “Selected Friends” the owner can specify a subset of his friends which gets access to the album and using “Email” access can be given to someone without a profile by sending him an Email. In addition to the sharing options in Table 4 Facebook also supports more advanced sharing settings which are described at the end of this section.

While the privacy settings for albums can be compared relatively easy it’s a bit harder for privacy settings for profile information. While doing a short examination of the possibilities with eight of the most important online social network sites allow to control the visibility of profile information, those two main concepts were observed: sharing of groups and hiding of individual items. Facebook, MySpace and StudiVZ use the concept of sharing groups, which means that the profile items are assigned to groups

such as “Basic Info”, “Contact Info”, “Interests”, “Friends” et cetera. Some of these groups have a fixed visibility and for the other groups the user can choose with whom he wants to share the fields of this group. The sharing settings from which the user can choose are for all sites except Facebook a subset of the privacy levels from Table 4. Facebook again allows more complex sharing levels which are described at the end of this section. Orkut, Hi5, Netlog, LinkedIn and Xing follow another approach: on those sites some profile items have a fixed visibility whereas the visibility of the other profile items can be restricted individually.

For both albums and profile information, Facebook supports more complex sharing rules than the other reviewed online social network sites. Those rules consist of a whitelist and a blacklist. Access is granted to anyone who is in the whitelist unless he’s also in the blacklist. The whitelist can be either one of “Friends of Friends”, “Only Friends”, “Only me” or a list of individual friends and friend lists. Using the blacklist one can exclude individual friends or friend lists from getting access.

4.2 Proposal for OpenSocial Access Control Lists

While designing a generic OpenSocial Access Control List which can be used to control access to different resources in different online social networks the two main issues which had to be solved were:

- define sharing model
- define entities one can share content with

The hardest problem while defining the sharing model was to find the correct trade-off between power and simplicity. It’s easy to define a simple API such as the MySpace extension to support album access control as described in Section 2.1.2. However such a simple API is very limited since not even friend lists can be used to share a resource. One could on the other hand also define an API which uses iptables like rules to determine whether someone can access a resource. Each rule could decide to allow, deny access or pass the decision to the next rule. If no rule matches, a default policy would apply. Such an API is very powerful but it comes with complexity as a drawback. If the sharing model of the online social network does not use a similar representation it’s hard to map from those rules to the online social network’s internal access control lists. The container’s sharing model is likely to be less powerful and therefore only a small set of the access control lists which are supported by this complex API can be used to represent valid sharing settings for that network and therefore the API must also provide information about which ACLs are valid for a given container which would make it even

more difficult to use such an API to modify ACLs. It is also difficult to explain such an ACL to the user which makes it hard to create a UI which can be understood by the average user. As described in Section 4.1 most current online social networks only allow for a whitelist which grants access to sets of users or external users via Email. The only exception which was found is Facebook which also supports to blacklist certain sets of users. Since Facebook is not likely to implement the OpenSocial standard and the average user is not likely to use such custom sharing settings a simpler ACL was preferred to a more powerful while still supporting more complex settings: the ACL consists of a whitelist which grants access to the resource to sets of users or external contacts. To support more complex settings which cannot be expressed using any of the sets of users proposed, a *custom* entry is defined which contains a textual description about the set and possibly also the number of users in that set. With this trade-off the API can be kept reasonably simple for cases which are considered to be common while still providing some information to the user in cases of more complex sharing settings.

To decide which entities should be supported to share items with, the results of the survey which are described in Section 4.1 were used to determine which entities are used to share items with in today's online social networks. This are the entities "Everybody", "All Users", "Friends of Friends", "Friends Only", "Me Only", "Selected Friends" and "Email" as defined in Section 4.1. "Family" is only supported by Flickr, which in not an OpenSocial container, but support for this entity was added because we expect that additional online social networks will support sharing with one's family in the future. "Everybody", "All Users", "Me Only" and "Family" are represented by the predefined OpenSocial groups named "@everybody", "@all", "@self" and "@family". "Friends Only" and "Friends of Friends" can be represented by the predefined group "@friends" which can be parametrized using a parameter which defines the network distance within which a user is considered to be a friend. Single users can be represented by their user IDs and multiple users which are members of an OpenSocial group using that group's ID. In addition to sharing via Email we also proposed to add support for sharing data with someone using that person's telephone number (for sharing by SMS). This might become useful in the future when more people connect to online social networks using their mobile phones. Those external contacts are being specified by the Email address and the phone number respective.

To provide a simple metric on how "public" or "private" an ACL's resource is, the server can provide the number of users who get access to the resource because of single entries and also the ACL as a whole. This metric can be computed on the server much more easily than on the client because the client would need to fetch all the sets whereas the server already has this information available. In some cases the online social network does not want to disclose the number of users for certain sets as for example

the number of registered users might be confidential. In this case the server can provide an approximation and mark it as such.

Since in OpenSocial all the more complex data types are objects consisting of fields, the proposed ACL contains also a list which specifies which fields the ACL applies to. This makes it possible to have different ACLs for different fields of an object. This is especially useful for privacy settings for profile items as described in Section 4.5. The proposal which contains the details of the proposed ACL can be found in Appendix A. This proposal was submitted to the OpenSocial community on February 28th 2010.

4.3 Proposal for OpenSocial Activities Privacy API

The proposal for an OpenSocial Activities Privacy API described in Appendix C is a good example to show how to use the proposed ACL to describe access control settings for a simple service. Only three small extensions are required. To attach the ACL to the activity, an array of ACLs is added to the activity data structure to hold all the ACLs attached to an activity. Multiple ACLs are attached such that different parts of the activity can have different ACLs. Since the ACLs are only meta data of the activity and therefore for many applications not of much use, developers might often not want to receive and send the ACLs. Therefore an optional boolean parameter was added to all the functions which retrieve or modify activities. This parameter can be used to explicitly request the ACLs when retrieving activities. In case of a function which modifies or creates an activity the added parameter specifies whether the activity sent by the client contains ACLs. The final change is to add a new function to retrieve the supported ACL entry types. This is needed when someone wants to modify an ACL to know which types of sharing are supported by the sharing model of the online social network. Some networks might for example not support to share data with someone who is not a member of the online social network.

4.4 Proposal for OpenSocial Albums and Media Items Privacy API

An API for access control to media items, such as photos, videos or audio files, is more complex than the API for activities because in addition to the media items there also exists the album which contains the media items. From this structure it follows that access control can happen on both levels. In today's online social networks access control happens only on the album level. This means that the album and all its media items have the same sharing settings. The one exception, found during this thesis, is the sharing model of Facebook which allows to share a single photo with someone by tagging him in that photo as described in Section 2.3. Since it can be expected that with additional

features other online social networks will do access control to photos on a per photo level, the API was designed to allow access control on the media item level. To allow that the media items and also the albums have ACLs. The ACL of the album is the default ACL for all the media items in that album which applies when the media item does not have an ACL attached itself. This way a client which wants to read the ACL does not need to distinguish between containers which support access control on the media item level and those which only support it on the album level. If the container supports access control only on the album level, no media item has an ACL attached and therefore the album's ACL specifies who gets access to the media item. Using this semantic it's important to distinguish between a media item with an empty ACL attached and a media item with no ACL attached. An empty ACL means that no one gets access to the media item whereas no ACL means that the album's ACL applies. Table 5 shows how access rights are determined for media items and albums based on the ACLs attached to the album and media item.

ACL	**not present**	**without entries**	**with entries**
album	ACL undefined	only owner has access	ACL applies
media item	album's ACL applies	only owner has access	ACL applies

Table 5: Interpreting ACLs for albums and media items

This model can be implemented by applying the proposed changes for the activities service to both the albums and the media items service and using the semantics from above to define how the ACLs are interpreted. The full proposal can be found in Appendix B.

4.5 OpenSocial Privacy API for Profile Fields

When it comes to sharing settings for profile items, every online social network has its own model. Some networks allow to specify the sharing level for single items whereas other networks assign profile items to groups and for each group the user can specify a sharing level. These groups, which will be denoted as *hidden groups*, are not important to read ACLs but become very important once one wants to modify the ACLs. If the hidden groups are not known to the client application it might try to set the sharing settings for the profile items to values which no longer satisfy the constraint that all the items in one hidden group have the same ACL. Since those ACLs cannot be mapped to a valid sharing setting supported by the server, the server has two options: reject the new settings (possibly with an error message) or adjust the requested settings according

to some rule such that the hidden group constraints are met. Both options are not particularly good ones. Rejecting inconsistent settings makes it hard to change any ACLs without knowing the hidden groups. Adjusting the settings to meet the constraints might have unpredictable and harmful effects because it might not result in the sharing setting which the user intended. Instead of adding additional meta information such that the hidden group constraints can be met the following approach can be used: for every hidden group an ACL is attached to the profile which represents the ACL for this hidden group. As long as only the ACL entries are modified and the fields to which the ACLs are assigned to stay the same the hidden group constraints are met.

5 Evaluation

5.1 Comparison to existing APIs

To the best of our knowledge, the proposed API is the first API to access user data across different online social networks which provides sharing controls. Once the proposed API is implemented by OpenSocial containers, it will allow for provider independent applications which provide the user with transparency of who has access to his data and control over which entities the data is shared with. This is not possible at the moment because if an online social network provides privacy features in its API those features are limited to this particular provider such that the application needs to use all these different APIs. Many online social networks do not provide such an API at all which will hopefully change once the proposed API is part of OpenSocial.

Compared to the APIs provided by Flickr and MySpace for privacy settings for photos, the proposed API is much more flexible. As it is to be expected from proprietary APIs, both APIs provide just the sharing settings which are supported by their web UIs: MySpace allows to choose between shared with no one, shared with friends only and shared with everyone with access to the internet. Flickr in addition also allows to share the photo with your family. Family and friends can be combined such that the photo is shared with friends and family. The proposed API on the other hand supports also sharing with groups which can consist of an arbitrary set of users. Sharing with individual online social network users and external users is also supported. One might argue that those proprietary APIs don't need to support sharing settings which are not supported by the sharing sharing model of the online social network because only the settings which are supported by the online social network can be used. However such limited APIs make it hard to modify the sharing model without breaking existing applications which use the old API. The Facebook API is less likely to cause this kind of problems because, as described in Section 2.1.1, its privacy part is much more flexible. Unfortunately, the Facebook API supports two formats for privacy settings. One allows for fine grained control whereas the other one is similar to the one used by MySpace and not able to represent all the sharing settings which are available in the web UI. Even worse, some functions support both formats whereas some other functions which operate on the same resources only support the limited format. One example for this is that when creating a new photo album one can use the complex setting whereas only the simple format is returned when one fetches the albums of a user. This makes it very hard to use the more complex privacy settings because these complex settings cannot be displayed to the user after retrieving the albums from the server. The API proposed in this thesis does not have this problem because all methods use the same format for privacy settings.

5.2 Photocial - Photo Sharing Application for Android

To support the proposals which extend OpenSocial with privacy features I wanted to create an example application to provide a use case to show that those API extensions enable for features to support the user in sharing his photos on online social networks in a comfortable and secure manner. I decided to write an Android application because Android phones often come with built in photo cameras. This and the fact that such phones are often connected to the internet make the process of taking a picture and uploading it to online social networks particularly convenient.

I defined the following main objectives for the application: it should be *transparent* with whom the photo is being shared and the user should have the necessary *controls* to choose with whom it is shared. This transparency and control should come without interfering with the process of the user uploading a photo. To achieve the desired transparency, control and simplicity the basic work flow is as follows: the user takes a photo and chooses to share it using Photocial. He then is presented with a screen showing the picture and the list of online social networks to which he is connected. For each network, the album name and its sharing setting is displayed (see Figure 1). By clicking on the network, the user can choose another album or create a new one. To simplify the switching between different sharing scenarios one can create profiles. For each of these profiles Photocial remembers the last used choice of albums to upload to.

While implementing Photocial I faced a few challenges: While the Android SDK provides neat APIs for many frequent tasks needed to write mobile applications, it also has some peculiarities which I, especially in the beginning, had to cope with. One of them is that Android by default destroys all the UI components when the phone configuration changes, which is for example the case when the screen is being rotated from portrait to landscape mode. After the configuration has been changed the UI will be recreated automatically . This means that when one has a background thread which performs some action and after finishing updates the UI, one needs to be really careful with keeping references to anything within the UI. When the background thread keeps references to old UI components, this might cause two potential problems: one problem is the increased memory usage because the garbage collector cannot collect those objects. This is a serious problem because on Android, memory is a limited resource and dealing with images can already cause exceptions because the process runs out of memory if one is not careful. The second and even more serious problem is that the background thread will update the UI which is no longer displayed and the currently displayed UI components will not be updated. To simplify creating background tasks which depend on having references to UI elements I created a base class which can be detached from the UI when the UI is being destroyed and reattached once the new UI was created. While the task

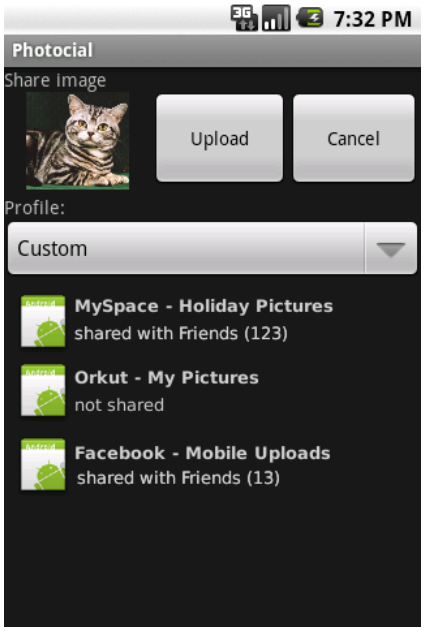

Figure 1: Photocial screen to share image

is not attached to any UI, operations to change the UI, are queued for execution when the task is attached to the new UI. Even though no references to the UI remain when the screen was rotated, the application still crashed from time to time when the screen was rotated. The problem was that garbage collection did not collect the unreachable objects before the preview image was being displayed and the process therefore run out of memory. Unfortunately, I did not find another solution to fix this other than to catch the exception, tell the system that it would be a good moment to do garbage collection and retry loading the image. Although this solution does not seem like a particularly good one it still fixed the problem. To communicate with OpenSocial containers there exists a Java library available at `http://code.google.com/p/opensocial-java-client/` which can be used on Android. The problem with this library was that when I started working on Photocial it was pretty outdated. Media items and albums for example were not yet supported. However during this time it was rewritten with a completely new design, so I decided to use the new, very experimental, version. To use the library I had to implement three-legged OAuth support and fix some bugs. Almost all of these changes have been included in the open source project.

Photocial now consists of about 5600 lines of Java code and more than 500 lines of XML files which contain the layout of the UI and the strings which would need to be

translated if one wanted to translate Photocial into another language. I plan to release Photocial as open source and distribute it on the Android Market.

5.3 Album Privacy API in Orkut

To add a new feature to OpenSocial it is generally expected that, in addition to sending a proposal to the OpenSocial and Gadgets Specification Discussion forum [17], also a proof of concept implementation is provided. To increase the chances that the proposed API changes will be included into the next version of the OpenSocial specification, the read only part of the albums API has been implemented as a part of the Orkut OpenSocial RPC endpoint.

After receiving a few hints at where the code is located and how to start a local Orkut server by Orkut developers, the changes which touched 9 files and added 4 new files were implemented. After some minor changes during the code review process the code was submitted to the Google code repository.

6 Conclusion

This Master's thesis consists of three major contributions:

- three proposals to extend OpenSocial
- an Android application to provide a use case for this extensions
- proof of concept implementation of the extensions in Orkut

The proposed changes to the OpenSocial specification constitute the first privacy features in OpenSocial once included in the standard. Those changes are split in three proposals: The first proposal introduces a generic access control list designed in an extensible way such that it can be reused for access control lists in many parts of OpenSocial. The two other proposals use this access control list to control access to albums/media items and activities. Photocial, the Android photo sharing application written during this thesis, represents a good use case to show how such an API can be used to protect the user's privacy when sharing pictures in online social networks. To make our proposal stronger we created a proof of concept implementation of our extensions in Google's online social network Orkut as required by the OpenSocial community

Future Work

1. Our proposal could be extended to cover privacy settings for profile fields. The tricky thing is the hidden constraints of fields that have common sharing settings and can't be changed independently. An approach like the one described in Section 4.5 could be used to address this.
2. The Android application could be extended to include activities, browsing of albums, privacy warnings etc.

A Proposal for OpenSocial Access Control Lists

Authors: Christoph Renner, Thomas Duebendorfer <api.duebendorfer@google.com>
Version of proposal document: 1.0 (March 24th, 2010)

This API proposal is compliant with the Open Social 1.0 API specification.

Acknowledgements to Mark Weitzel from IBM, Rahul Kulkarni and Sachin Shenoy from Orkut, Lane LiaBraaten from Google, Tyrone W. Grandison from IBM Almaden and Martin Burkhart from ETH Zurich for their feedback.

A.1 Motivation

Sharing information is one of the most important features of online social networks. Such shared information is often personal and only intended to be shared with certain users or groups of users. Unintended leakage of personal information can result in damage of someone's reputation, identity theft or can have other bad effects for the user. To share information in a comfortable and secure way the user needs to know with whom personal data is shared and must be able to restrict who can access it. Many online social networks provide such functionality in their Web UIs. This proposal will add privacy controls to OpenSocial by defining syntax and semantics of access control lists and operations on them. Providing privacy controls across multiple social network services in a standardized way makes it easier for developers to incorporate privacy friendly features in their applications, which will give the user better transparency of and control over personal information.

A.2 Overview

This proposal extends the OpenSocial Social Data Specification 1.0 with an ACL data object that can be used in resources to support access control lists.

A.3 Changes to the Social Data Specification

Acl

Add new Acl data object under Additional Social Data.

An ACL is always attached to a resource or a part of a resource which has an owner. By

default every ACL contains an implicit entry for the owner which can be overridden by adding an explicit ACL entry for the owner.

Field Name	**Field Type**	**Description**
entries	Array<Acl-Entry>	The ACL grants subjects access to the object, which this acl is attached to, according to the entries in this array.
numberOfPeople	Number-Of-People	Optional. The number of people who get access because of this ACL. This field helps to indicate to the user how "private" or "public" a shared item is. It's more efficient to compute this on the server than on the client. The server must ignore this field in requests from the client.
fields	Array<string>	Optional. If an ACL applies only to certain fields of a resource, this array contains the field names of those fields. If fields is not set or empty the ACL applies to all fields except for those for which an explicit ACL exists. A resource cannot have multiple ACLs for the same field. This field cannot be changed by the client. The server must ignore this field in requests from the client.

Acl-Entry

Add new Acl-Entry data object under Additional Social Data.

An Acl-Entry grants access to a set of subjects. The following table describes valid fields for Acl-Entries.

Field Name	Field Type	Description
type	Acl-Entry-Type	The type of this ACL entry. This field must always be set to one of the predefined values.
accessorId	string	An accessor ID where applicable. The following list describes the meaning depending on the type: **USER**: The User-ID. **GROUP**: The Group-ID. To indicate a group with just the owner, use @self. **EXTERNAL_CONTACT**: The contact, see table "Accessor Types" for valid values. This field must be set if type is GROUP, USER or EXTERNAL_CONTACT.
accessorType	Accessor-Type	Specifies the accessor type if the Acl-Entry-Type is not specific enough. This field must be set if type equals EXTERNAL_CONTACT. For all other types it is optional.
accessorRights	Array<Accessor-Right>	Optional. Default: ["GET", "POST", "PUT", "DELETE"] for the implicit owner in the ACL for the whole resource (fields not set), ["GET", "PUT"] for the implicit owner in field specifyc ACLs and ["GET"] otherwise. "DELETE" and "POST" can only be used in ACLs for the whole resource because these operations can only be performed on the resource as a whole.
description	string	A human readable description of the people to whom this entry grants access. This field must be set if type equals CUSTOM.
numberOfPeople	Number-Of-People	Optional. The number of people who get access because of this ACL entry. The server must ignore this field in requests from the client.
networkDistance	integer	Optional. Containers MAY support the network distance field, which modifies (@friends, etc.) to include the transitive closure of all friends up to the specified distance away.

Acl-Entry-Type

Add new Acl-Entry-Type data object under Additional Social Data.

Specifies the type of an ACL entry.

```
Acl-Entry-Type = "GROUP" / "USER" / "EXTERNAL_CONTACT" / "CUSTOM"
```

Those values are described in the following table and expressed from the viewpoint of the owner of the resource.

Acl-Entry-Type	**description**
GROUP	A group as defined in OpenSocial 1.0.
USER	A single social network service user.
EXTERNAL_CONTACT	A contact which is not a user of the social network service.
CUSTOM	This value must only be used if none of the above is appropriate. It indicates a proprietary extension. Standard compliant clients can show the description to the user but the interpretation of all the other fields, including additional proprietary fields, depends on the container's implementation.

Group ID

Change the Group ID section to the following:

The group ID must only contain alphanumeric (A-Za-z0-9) characters, underscore (_), dot(.) or dash(-), and must uniquely identify the group in a container.

```
Group-ID = Object-Id / "@self" / "@friends" / "@all" / "@everybody" /
    "@family"
```

The following table describes the predefined Group-IDs.

Group-ID	**description**
@self	When used in the context of a user, this group contains only that user.
@friends	Contains all the user's friends.
@all	Contains all the users in that container.
@everybody	Contains all Internet users.
@family	Contains all the users which belong to a user's family.

Accessor-Type

Add new Accessor-Type data object under Additional Social Data.

Specifies the type of accessor more precisely than the type field. For Acl-Entry-Type EXTERNAL_CONTACT, Table 6 lists a few proposed values. Other values can be used as a proprietary extension using a prefix starting with a lowercase letter.

```
Accessor-Type = "MAILTO" / "PHONE" / Custom-Accessor-Type
Custom-Accessor-Type = LOALPHA TEXT
```

Accessor-Type	**description**	**accessorId**
MAILTO	Shared via email e.g. by sending a link.	an email address
PHONE	Shared with someone who is specified by his phone number e.g. by sending a text message.	a phone number

Table 6: Accessor Types

Accessor-Right

Add new Accessor-Right data object under Additional Social Data.

```
Accessor-Right = "GET" / "POST" / "PUT" / "DELETE"
```

Grants the permission to perform the same-named REST method or its corresponding RPC call on the resource. This means that "GET" gives read access, "PUT" allows to modify fields of the resource (including removing the field), "DELETE" grants the right to delete the resource and "POST" allows to create new resources.

Number-Of-People

Add new Number-Of-People data object under Additional Social Data.

Field Name	**Field Type**	**Description**
count	integer	An integer describing the number of people. Can be an approximation in which case isApproximate needs to be set to true.
isApproximate	Boolean	Optional. True if count is not an exact number but an estimate. Defaults to "false".

Supported Acl Entry Type

Add new Supported-Acl-Entry-Type data object under Additional Social Data.

Field Name	Field Type	Description
type	Acl-Entry-Type	A supported Acl-Entry-Type.
accessorId	Array<Group-ID>	Optional. If type is GROUP this field contains predefined Group-ID values (@friends, @family, etc) which are supported to be used as accessor IDs.
accessorType	Array<Accesor-Type>	Optional. If type is EXTERNAL_CONTACT this field contains supported accessorType values.

A.4 Examples

Shared with all friends, the user with ID example.tld.user.1234

XML

```
<acls>
  <entries>
    <type>GROUP</type>
    <accessorId>@friends</accessorId>
    <numberOfPeople><count>125</count></numberOfPeople>
  </entries>
  <entries>
    <type>USER</type>
    <numberOfPeople><count>1</count></numberOfPeople>
    <accessorId>example.org.user.1234</accessorId>
  </entries>
  <numberOfPeople><count>126</count></numberOfPeople>
</acls>
```

JSON

```
"acls": [{
  "entries": [{
    "type": "GROUP",
    "accessorId": "@friends",
```

```
    "numberOfPeople": {"count": 125}
  },{
    "type": "USER",
    "numberOfPeople": {"count": 1},
    "accessorId": "example.org.user.1234"
  }],
  "numberOfPeople": {"count": 126}
}]
```

Resource is shared with everyone except field “email” which is kept private. Since the second ACL contains no entries it grants access only to the implicit owner.

XML

```
<acls>
  <entries>
    <type>GROUP</type>
    <accessorId>@all</accessorId>
    <numberOfPeople>
      <count>1000000</count>
      <isApproximate>true</isApproximate>
    </numberOfPeople>
  </entries>
</acls>
<acls>
  <fields>email</fields>
</acls>
```

JSON

```
"acls": [{
  "entries": [{
    "type": "GROUP",
    "accessorId": "@all",
    "numberOfPeople": {
      "count": 1000000,
      "isApproximate": true
    }
  }],
  "numberOfPeople": {
```

```
    "count": 1000000,
    "isApproximate": true
  }
}, {
  "fields": ["email"]
}]
```

Shared with all users and someone which is not a user of the social network

XML

```
<acls>
  <entries>
    <type>GROUP</type>
    <accessorId>@all</accessorId>
    <numberOfPeople>
      <count>1000000</count>
      <isApproximate>true</isApproximate>
    </numberOfPeople>
  </entries>
  <entries>
    <type>EXTERNAL_CONTACT</type>
    <accessorType>MAILTO</accessorType>
    <numberOfPeople><count>1</count></numberOfPeople>
    <accessorId>joe@mailhost.tld</accessorId>
  </entries>
  <numberOfPeople>
    <count>1000000</count>
    <isApproximate>true</isApproximate>
  </numberOfPeople>
</acls>
```

JSON

```
"acls": [{
  "entries": [{
    "type": "GROUP",
    "accessorId": "@all",
    "numberOfPeople": {
      "count": 1000000,
```

```
      "isApproximate": true
    }
  }, {
    "type": "EXTERNAL_CONTACT",
    "accessorType": "MAILTO",
    "accessorId": "joe@mailhost.tld",
    "numberOfPeople": {"count": 1}
  }],
  "numberOfPeople": {
    "count": 1000000,
    "isApproximate": true
  }
}]
```

B OpenSocial Album/Media Item Privacy API Proposal

Authors: Christoph Renner, Thomas Duebendorfer <api.duebendorfer@google.com>

Version of proposal document: 1.0 (March 24th, 2010)

This API proposal is compliant with the Open Social 1.0 API specification.

Acknowledgements to Rahul Kulkarni, Sachin Shenoy from Orkut, Lane LiaBraaten from Google, Tyrone W. Grandison from IBM Almaden and Martin Burkhart from ETH Zurich for their feedback.

B.1 Motivation

Sharing of media items such as photos or videos has become an important feature of many social network services. Sharing media items is an important privacy issue because they can reveal a lot of personal information to recruiters, insurance companies and others who might use this information negatively. While many social network services have implemented media item access control functionality in their Web UI or in a proprietary API, currently no such access controls exist in OpenSocial. Providing privacy controls across multiple social network services in a standardized way makes it easier for developers to incorporate privacy features in their applications, which will give the user better transparency of and control over personal information.

B.2 Overview

This proposal extends the MediaItems and Albums Service as defined in the OpenSocial 1.0 specification to retrieve and modify access control settings for media items and albums. It builds on the proposal "Proposal for an OpenSocial ACL".

B.3 Semantics of access control lists

Our proposal is to add a new field called "acl" (access control lists) to the Album data object. In addition, we propose to add the same field to the MediaItem data object, which allows one to use different access control lists for media items in an album. All media items in an album share the same ACLs as the album by default unless an ACL is returned in a MediaItem response.

The following table describes the meaning of missing, empty and non empty acl fields for albums and media items.

ACL	**not present**	**without entries**	**with entries**
album	ACL undefined	only owner has access	ACL applies
media item	album's ACL applies	only owner has access	ACL applies

For example, if the album is shared with everyone and the media item's has an ACL but contains no entries, then only the owner can access that media item. If the album's ACL contains no entries and a media item in that album is shared with someone that person gets access only to this media item but not to the whole album.

B.4 Changes to the Social Data Specification

Album

Add new optional field acl to Album.

Field Name	**Field Type**	**Description**
acl	Array<Acl>	Optional Access control lists which describes access to that album and the default access for media items in that album.

MediaItem

Add new optional field acl to MediaItem.

Field Name	**Field Type**	**Description**
acl	Array<Acl>	Optional. Access control lists which grants access to that media item. When this field is present access is only granted to all the subjects for which an entry in one of these ACLs exists.

B.5 Changes to the Social API Server Specification

Get Albums Request Parameters

Name	Type	Description
acl	Boolean	Specifies whether to include the ACLs in the response. Defaults to “false”.

Update Album Request Parameters

Name	Type	Description
acl	Boolean	Specifies whether the request intends to change the ACLs. When set to “true” and the Album in the data parameter does not contain an acl field, the container MUST set the Album’s ACLs to a reasonable default value. Defaults to “false”.

Get MediaItems Request Parameters

Name	Type	Description
acl	Boolean	Specifies whether to include the ACLs in the response. Defaults to “false”.

Update MediaItem Request Parameters

Name	Type	Description
acl	Boolean	Specifies whether the request intends to change the ACLs. When set to “true” and the MediaItem in the data parameter does not contain an acl field, the container MUST delete that media item’s ACLs such that the media item inherits the ACLs of its album. Defaults to “false”.

Retrieve a list of supported ACL entry types for albums

Containers MAY support requests for a list of supported ACL-Type values for albums. Requests and responses use the following values:

```
REST-HTTP-Method       = "GET"
```

```
REST-URI-Fragment      = "/albums/@supportedAclEntryTypes"
REST-Query-Parameters  = null
REST-Request-Payload   = null

RPC-Method             = "albums.getSupportedAclEntryTypes"
RPC-Request-Parameters = null

Return-Object          = Array<Supported-Acl-Entry-Type>
```

Retrieve a list of supported ACL entry types for media items

Containers MAY support requests for a list of supported ACL entry types for media items. If setting ACLs per media item is not supported the container should return an empty array. Requests and responses use the following values:

```
REST-HTTP-Method       = "GET"
REST-URI-Fragment      = "/mediaItems/@supportedAclEntryTypes"
REST-Query-Parameters  = null
REST-Request-Payload   = null

RPC-Method             = "mediaItems.getSupportedAclEntryTypes"
RPC-Request-Parameters = null

Return-Object          = Array<Supported-Acl-Entry-Type>
```

B.6 Changes to the Gadget / JavaScript API Specification

osapi.albums.getSupportedAclEntryTypes

Signature
<static> { osapi.Request } osapi.albums.getSupportedAclEntryTypes()

Description
Builds a request to retrieve all supported ACL entry types for albums.

Parameters
This method takes no parameters.

Returns

An osapi.Request to retrieve all supported ACL entry types for albums. Executing this request MUST return an array<Supported-Acl-Entry-Type>.

osapi.mediaItems.getSupportedAclEntryTypes

Signature
<static> { osapi.Request } osapi.mediaItems.getSupportedAclEntryTypes()

Description
Builds a request to retrieve all supported ACL entry types for media items.

Parameters
This method takes no parameters.

Returns
An osapi.Request to retrieve all supported ACL entry types for media items. Executing this request MUST return an array<Supported-Acl-Entry-Type>.

B.7 Examples

Requesting albums with access control lists

A request for the user's album list, which contains two albums with the following ACLs:

- First Album
  - Shared with all friends
  - Shared with user example.org.user.1234
- Second Album
  - Shared with all users of that social network service. That service does not want to disclose the exact number of users.
  - Shared with external contact joe@mailhost.tld via email.

REST request

```
GET /albums/@me/@self?acl=true
```

REST response

```
<response xmlns="http://ns.opensocial.org/2008/opensocial">
  <startIndex>1</startIndex>
  <itemsPerPage>10</itemsPerPage>
  <totalResults>2</totalResults>
     <entry>
    <caption>First Album</caption>
    <id>example.org.album.123456</id>
    <thumbnailUrl>
      http://example.org/images/album_123456.png
    </thumbnailUrl>
    <acl>
      <entries>
        <type>GROUP</type>
        <accessorId>@friends</accessorId>
        <numberOfPeople><count>125</count></numberOfPeople>
      </entries>
      <entries>
        <type>USER</type>
        <numberOfPeople><count>1</count></numberOfPeople>
        <accessorId>example.org.user.1234</accessorId>
      </entries>
      <numberOfPeople><count>126</count></numberOfPeople>
    </acl>
  </entry>
  <entry>
    <caption>Second Album</caption>
    <id>example.org.album.654321</id>
    <thumbnailUrl>
      http://example.org/images/album_654321.png
    </thumbnailUrl>
    <acl>
      <entries>
        <type>GROUP</type>
        <accessorId>@all</accessorId>
        <numberOfPeople>
          <count>1000000</count>
          <isApproximate>true</isApproximate>
        </numberOfPeople>
      </entries>
      <entries>
```

```
        <type>EXTERNAL_CONTACT</type>
        <accessorType>MAILTO</accessorType>
        <numberOfPeople><count>1</count></numberOfPeople>
        <accessorId>joe@mailhost.tld</accessorId>
      </entries>
      <numberOfPeople>
        <count>1000000</count>
        <isApproximate>true</isApproximate>
      </numberOfPeople>
    </acl>
  </entry>
</response>
```

JSON-RPC request

```
{
  "method": "albums.get",
  "id": "myalbums",
  "params": {
    "userId": "@me",
    "groupId": "@self",
    "acl": "true"
  }
}
```

JSON-RPC response

```
{
  "id": "myalbums",
  "result": {
    "startIndex": 0,
    "itemsPerPage": 10,
    "totalResults": 2,
    "list": [{
      "caption": "First Album",
      "id": "example.org.album.123456",
      "thumbnailUrl": "http:\/\/example.org/images/album_123456.png",
      "acl": [{
        "entries": [{
```

```
          "type": "GROUP",
          "accessorId": "@friends",
          "numberOfPeople": {"count": 125}
        },{
          "type": "USER",
          "numberOfPeople": {"count": 1},
          "accessorId": "example.org.user.1234"
        }],
        "numberOfPeople": {"count": 126}
      }]
    }, {
      "caption": "Second Album",
      "id": "example.org.album.654321",
      "thumbnailUrl": "http:\/\/example.org/images/album_654321.png",
      "acl": [{
        "entries": [{
          "type": "GROUP",
          "accessorId": "@all",
          "numberOfPeople": {
            "count": 1000000,
            "isApproximate": true
          }
        }, {
          "type": "EXTERNAL_CONTACT",
          "accessorType": "MAILTO",
          "accessorId": "joe@mailhost.tld",
          "numberOfPeople": {"count": 1}
        }],
        "numberOfPeople": {
          "count": 1000000,
          "isApproximate": true
        }
    }]]
  }
}
```


Request a single media item with ACL

A sample media item shared with friends and friends of friends.

REST request

```
GET /mediaItems/@me/@self/example.org.album.123456/
    example.org.image.1234?acl=true
```

REST response

```
<mediaitem xmlns="http://ns.opensocial.org/2008/opensocial">
  <id>example.org.image.1234</id>
  <thumbnailUrl>http://example.org/images/1234-tn.png</thumbnailUrl>
  <type>image</type>
  <url>http://example.org/images/1234.png</url>
  <albumId>example.org.album.123456</albumId>
  <acl>
    <entries>
      <type>GROUP</type>
      <accessorId>@friends</accessorId>
      <networkDistance>2</networkDistance>
      <numberOfPeople><count>589</count></numberOfPeople>
    </entries>
  </acl>
</mediaitem>
```

JSON-RPC request

```
{
  "method": "mediaItems.get",
  "id": "firstalbum",
  "params": {
    "id": ["example.org.image.1234"],
    "acl": "true"
  }
}
```

JSON-RPC response

```
{
  "id": "firstalbum",
  "result": {
```

```
    "id": "example.org.image.1234",
    "thumbnailUrl": "http:\/\/example.org\/images\/1234-tn.png",
    "type": "image",
    "url": "http:\/\/example.org\/images\/1234.png",
    "albumId": "example.org.album.123456",
    "acl": [{
      "entries": [{
        "type": "GROUP",
        "accessorId": "@friends",
        "networkDistance": 2,
        "numberOfPeople": {"count": 589}
      }],
      "numberOfPeople": {"count": 589}
    }]
  }
}
```

Delete a media item's ACLs

Deletes the ACLs of a media item. As a consequence, the ACLs from the album will be inherited by default.

REST request

```
PUT /mediaItems/@me/@self/example.org.album.123456/
    example.org.image.1234?acl=true

<mediaitem xmlns="http://ns.opensocial.org/2008/opensocial">
  <id>example.org.image.1234</id>
  <thumbnailUrl>http://example.org/images/1234-tn.png</thumbnailUrl>
  <type>image</type>
  <url>http://example.org/images/1234.png</url>
  <albumId>example.org.album.123456</albumId>
</mediaitem>
```

JSON-RPC request

```
{
  "method": "mediaItems.update",
```

```
  "id": "empty list",
  "params": {
    "id": ["example.org.image.1234"],
    "acl": "true",
    "data": {
      "id": "example.org.image.1234",
      "thumbnailUrl": "http:\/\/example.org\/images\/1234-tn.png",
      "type": "image",
      "url": "http:\/\/example.org\/images\/1234.png",
      "albumId": "example.org.album.123456"
    }
  }
}
```

Requesting supported ACL entry types for albums

REST request

```
GET /albums/@supportedAclEntryTypes
```

REST response

```
<supportedaclentrytype>
  <type>USER</type>
</supportedaclentrytype>
<supportedaclentrytype>
  <type>GROUP</type>
  <accessorId>@friends</accessorId>
  <accessorId>@all</accessorId>
  <accessorId>@self</accessorId>
</supportedaclentrytype>
<supportedaclentrytype>
  <type>EXTERNAL_CONTACT</type>
  <accessorType>MAILTO</accessorType>
</supportedaclentrytype>
```

JSON-RPC request

```
{
```

```
  "method": "albums.getSupportedAclEntryTypes",
  "id": "someid"
}
```

JSON-RPC response

```
{
  "id": "someid",
  "result": [{
    "type": "USER"
  }, {
    "type": "GROUP",
    "accessorId": ["@friends", "@all", "@self"]
  }, {
    "type": "EXTERNAL_CONTACT",
    "accessorType": ["MAILTO"]
  }]
}
```

C Proposal for OpenSocial Activities Privacy API

Authors: Christoph Renner, Thomas Duebendorfer <api.duebendorfer@google.com>

Version of proposal document: 0.2 (March 24th, 2010)

This API proposal is compliant with the Open Social 1.0 API specification.

C.1 Overview

This proposal extends the Activities service as defined in the OpenSocial 1.0 specification to retrieve and modify access control settings. It builds on the proposal "Proposal for an OpenSocial ACL".

C.2 Activity Stream Access Control in Social Network Services

Although activities have the potential to reveal privacy sensitive information, the OpenSocial 1.0 API does not allow to control to whom an activity will be shown on an activity basis. This feature is not widely available in online social network's WebUIs either. The only provider we found to support that feature is Facebook which supports fine grained access control settings for individual status updates, links and other posts. Even though it is quite early to define a standard before this feature is available in several major online social networks, it would be nice to support that functionality by including it in the OpenSocial specification. In addition to the control that this proposal gives to the user, it also enables the application developers to be more transparent about to whom they reveal information by posting to the user's activities.

C.3 Changes to the Social Data Specification

Activity

Add new optional field acl to Activity.

Field Name	Field Type	Description
acl	Array<Acl>	Optional Access control lists which grant access to that activity.

C.4 Changes to the Social Server API Specification

Get Activities Request Parameters

Add new parameter 'acl'.

Name	Type	Description
acl	Boolean	Specifies whether to include the ACLs in the response. Defaults to "false".

Update Activities Request Parameters

Add new parameter 'acl'.

Name	Type	Description
acl	Boolean	Specifies whether the request intends to change the ACL. When set to "true" and the Activity in the activity parameter does not contain an acl field, the container MUST set the Activity's ACL to a reasonable default value. Defaults to "false".

Retrieve a list of supported ACL entry types for activities

Containers MAY support requests for a list of supported ACL-Type values for activities. Requests and responses use the following values:

```
REST-HTTP-Method       = "GET"
REST-URI-Fragment      = "/activities/@supportedAclEntryTypes"
REST-Query-Parameters  = null
REST-Request-Payload   = null

RPC-Method             = "activities.getSupportedAclEntryTypes"
RPC-Request-Parameters = null

Return-Object          = Array<Supported-Acl-Entry-Type>
```

C.5 Changes to the Gadget / JavaScript API Specification

osapi.activities.getSupportedAclEntryTypes

Signature
<static> { osapi.Request } osapi.activities.getSupportedAclEntryTypes()

Description
Builds a request to retrieve all supported ACL entry types for activities.

Parameters
This method takes no parameters.

Returns
An osapi.Request to retrieve all supported ACL entry types for activities. Executing this request MUST return an array¡Supported-Acl-Entry-Type¿.

C.6 Examples

Create an activity with ACL

Create an activity which is only visible to friends.

REST request

```
POST /activities/@me/@self

<activity xmlns="http://ns.opensocial.org/2008/opensocial">
  <title>some activity</title>
  <body>activity details</body>
  <acl>
    <entries>
      <type>GROUP</type>
      <accessorId>@friends</accessorId>
    </entries>
  </acl>
</activity>
```

JSON-RPC request

```
{
```

```
  "method": "activities.create",
  "id": "key",
  "params": {
    "userId": "@me",
    "groupId": "@self",
    "activity": {
      "title": "some activity",
      "body": "activity details",
      "acl": [{
        "entries": [{
          "type": "GROUP",
          "accessorId": "@friends"
        }]
      }]
    }
  }
}
```

Retrieve activities with ACL

Request the viewers activities. The first activity can be seen by all users of the container whereas the second activity is only visible to friends.

REST request

```
GET /activities/@me/@self?acl=true
```

REST response

```
<response xmlns="http://ns.opensocial.org/2008/opensocial">
  <startIndex>1</startIndex>
  <itemsPerPage>2</itemsPerPage>
  <totalResults>2</totalResults>
  <entry>
    <id>example.org.activity.1</id>
    <title>first activity</title>
    <acl>
      <entries>
        <type>GROUP</type>
```

```
        <accessorId>@all</accessorId>
      </entries>
    </acl>
  </entry>
  <entry>
    <id>example.org.activity.2</id>
    <title>second activity</title>
    <acl>
      <entries>
        <type>GROUP</type>
        <accessorId>@friends</accessorId>
      </entries>
    </acl>
  </entry>
</response>
```

JSON-RPC request

```
{
  "method": "activities.get",
  "id": "my_activities",
  "params": {
    "acl": "true"
  }
}
```

JSON-RPC response

```
{
  "id": "my_activities",
  "result": {
    "startIndex": 1,
    "itemsPerPage": 2,
    "totalResults": 2,
    "entry": [
      {
        "id": "example.org.activity.1",
        "title": "first activity",
        "acl": [{
```

```
          "entries": [{
            "type": "GROUP",
            "accessorId": "@all"
          }]
        }]
      }, {
        "id": "example.org.activity.2",
        "title": "second activity",
             "acl": [{
          "entries": [{
            "type": "GROUP",
            "accessorId": "@friends"
          }]
        }]
      }
    ]
  }
}
```

References

[1] Opensocial is 1 and reach is 600 million users at 20 sites. `http://blogs.sun.com/socialsite/entry/opensocial_is_1`.

[2] Please rob me. `http://pleaserobme.com/`.

[3] Alessandro Acquisti and Ralph Gross. Predicting social security numbers from public data. *Proceedings of the National Academy of Sciences*, 106(27):10975–10980, July 2009.

[4] Lars Backstrom, Cynthia Dwork, and Jon Kleinberg. Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 181–190, New York, NY, USA, 2007. ACM.

[5] Joseph Bonneau and Sören Preibusch. The privacy jungle: On the market for data protection in social networks. In *The Eighth Workshop on the Economics of Information Security (WEIS 2009)*, 2009.

[6] IBM Almaden Research Center. Privacy-aware market place facebook application. `http://apps.facebook.com/p_a_m_p`.

[7] Leucio Antonio Cutillo, Refik Molva, and Thorsten Strufe. Safebook: Feasibility of transitive cooperation for privacy on a decentralized social network. In *10th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, WOWMOM 2009, Kos Island, Greece, 15-19 June, 2009*, pages 1–6, 2009.

[8] Facebook. Facebook help centre. `http://www.facebook.com/help/`.

[9] Facebook. Facebook statistics. `http://www.facebook.com/press/info.php?statistics`, 2010.

[10] Google. Google launches opensocial to spread social applications across the web. `http://www.google.com/intl/en/press/pressrel/opensocial.html`, 2007.

[11] Collin Jackson, Andrew Bortz, Dan Boneh, and John C. Mitchell. Protecting browser state from web privacy attacks. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 737–744, New York, NY, USA, 2006. ACM.

[12] Kun Liu and Evimaria Terzi. Towards identity anonymization on graphs. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 93–106, New York, NY, USA, 2008. ACM.

[13] Kun Liu and Evimaria Terzi. A framework for computing the privacy scores of users in online social networks. In *ICDM '09: Proceedings of the 2009 Ninth IEEE International Conference on Data Mining*, pages 288–297, Washington, DC, USA, 2009. IEEE Computer Society.

[14] E. Michael Maximilien, Tyrone Grandison, Kun Liu, Tony Sun, Dwayne Richardson, and Sherry Guo. Enabling privacy as a fundamental construct for social networks. In *Proceedings IEEE CSE'09, 12th IEEE International Conference on Computational Science and Engineering, August 29-31, 2009, Vancouver, BC, Canada*, pages 1015–1020, 2009.

[15] MySpace. Opensocial v0.9 albums - myspace open platform: Documentation wiki. `http://wiki.developer.myspace.com/index.php?title=OpenSocial_v0.9_Albums#Notes`, 2010.

[16] NY Daily News. Nathalie blanchard loses benefits over facebook beach photos. `http://www.nydailynews.com/news/world/2009/11/22/2009-11-22_nathalie_blanchard_loses_benefits_over_facebook_beach_photos.html`.

[17] OpenSocial. Opensocial and gadgets specification discussion. `http://groups.google.com/group/opensocial-and-gadgets-spec`.

[18] OpenSocial. Opensocial specs. `http://www.opensocial.org/specs`, 2010.

[19] OpenSocial. Opensocial website. `http://www.opensocial.org`, 2010.

[20] World Wide Web Consortium (W3C). Platform for privacy preferences (p3p) project. `http://www.w3.org/P3P/`.

[21] Gilbert Wondracek, Thorsten Holz, Engin Kirda, and Kruegel Christopher. A practical attack to de-anonymize social network users. Technical report, International Secure Systems Lab, 2010.