Improving the Safety and Practicality of Authorization Technology Using LOAF

Takumi Akahoshi†, Manabu Okamoto‡, Michio Shimomura‡ and Naohisa Komatsu†

†Department of Computer Science and Engineering, Waseda University, 3-4-1 Okubo Shinjuku-ku, Tokyo, 169-8555 Japan
{akahoshi, komatsu}@kom.comm.waseda.ac.jp

‡NTT Information Sharing Platform Laboratories, NTT Corporation
3-9-11 Midoricho, Musashino-shi, Tokyo 180-8585, Japan
{shimomura.michio, okamoto.m}@lab.ntt.co.jp

ABSTRACT
Social Networking Services (SNS) such as Facebook in the U.S.A. and mixi in Japan are gathering much attention, and are growing to include huge numbers of users. Now, SNS are starting to require more detailed types of function in addition to direct, “friend” access control, such as “friend-of-a-friend”. Accordingly, we have completed a study on implementations of such “friend-of-a-friend” access control efficiently using LOAF (List-Of-All-Friends). We also examined the practicality and safety of a system that provides such access control, even across different systems.

Categories and Subject Descriptors

General Terms
Security

Keywords
Authentication, Identity Management, Access Control, OpenID

1. INTRODUCTION
Currently, Social Networking Services (SNS) are a type of user-participatory service that are gathering much attention, and have grown to include a huge number of users. One of the features of SNS often mentioned is that “friend” relationships between users can be used for access control. Currently, the range of user access requests is expanding for services and users are beginning to want more detailed functions. One such new function is access for “friends-of-friends” in addition to access for direct “friends”.

Accordingly, in this paper, we have investigated use of OpenID as a user ID, with its "global ID" property, for implementing "friend-of-a-friend" access control. We propose an effective method for accomplishing this using LOAF, and discuss important issues related to practicality and safety which came to light in this study.

2. OPENID
2.1 OVERVIEW
OpenID is a type of proxy authentication method. It makes use of a user ID coded in URI or XRI format, and is designed for implementing single-sign-on (SSO) systems[1].

By creating a single OpenID, users are able to log in easily to all web sites that support OpenID.

2.2 OpenID Authentication algorithm
The user enters an OpenID at a web site supporting OpenID (called a Relying Party, or RP). From the entered OpenID, the RP retrieves the IDs OP Endpoint URL, which is the address of the corresponding OpenID authentication Provider (OP). The RP redirects the user's browser to the OP Endpoint URL. OP authenticates the user and redirects the user's browser to the RP. Login is complete if authentication result is correct.

In the following sections, we propose use of OpenID for friend-of-a-friend access control and a method for implementing it.

3. Friend of a friend authorization model
3.1 Approval using white lists
A commonly mentioned feature of SNS is that they are able to use “friend” relationships between users for access control. One possible extension to the access control patterns available is to allow FOAF (friend-of-a-friend) relationships to be used. This is actually allowed on mixi. Currently, such access controls are limited to a single SNS, but in the future, we expect it will be necessary to perform access control spanning IDs from different SNS as well.

As such, we will first discuss the basic approach necessary to allow access control spanning IDs on different SNS through use of OpenID as the user ID. An overview of this is shown in Figure 1, and the processes for this method are described below.

①User A obtains an ID(v) which is OpenID of accessing party V, and sends the ID(v) to the RP.
②A request to authorize the accessing party V is sent to the user A’s OP.
③Check whether the accessing party’s ID(v) is in the friend list in the OP.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

DIM’09, November 13, 2009, Chicago, Illinois, USA.
Copyright 2009 ACM 978-1-60558-786-8/09/11...$10.00.
① If the ID(v) is not in the list, the party is not a direct friend, so check whether it is a friend of a friend.
② Based on the URL in the friend’s OpenID, send a friend authentication request to the friend’s OP.
③ If the accessing party v is a friend of a friend, accept access. If not, deny access.
④ Send a request for a LOAF to the accessing party V’s OP, and retrieve the LOAF.
⑤ Check whether any of the IDs in the user’s friend list are in the received LOAF.
⑥ If any ID of a user’s friend is in the LOAF, send a friend confirmation request to the OP of that friend.

(3) Check whether the accessing party V is in the user's friend list at the OP. If not in the list, the party is not a direct friend, so proceed to checking whether he/she is a friend-of-a-friend.
(4) Send a request for a LOAF to the accessing party V’s OP, and retrieve the LOAF.
(5) Check whether any of the IDs in the user’s friend list are in the received LOAF.
(6) If any ID of a user's friend is in the LOAF, send a friend confirmation request to the OP of that friend.

(7) If the friend has the accessing party V as a friend, allow access. If none of the user's friends has the accessing party as a friend, deny access.

Figure 1. FOAF authorization model

With the above procedure, it is possible to extend white-list access control functions to different SNS and to support friend-of-a-friend access control. However, this approach will cause many requests to friends as well as from friends, so a load of the system will be heavy.

In our research, we apply LOAF (list-of-all-friends) [4] in addition to white lists in order to solve above problem. LOAF is described in detail in the next section.

3.2 Overview of LOAF

In order to make the FOAF authorization model described in the previous section practical, we apply the LOAF of the users in question. The LOAF is represented in Bloom Filter form, which has the following features [5]:
- It preserves anonymity.
- It requires few resources and does not affect overall systems
- It can produce false positives

The Bloom filter, conceived by Burton H. Bloom in 1970, is a space-efficient probabilistic data structure that is used to test whether an element is a member of a set. False positives are possible, but false negatives are not [6].

4. Proposed method

4.1 Authorization model using LOAF

To make the friend-of-a-friend authorization model described in section 3.1 practical, we applied the LOAF to the white list. It is assumed that all Bloom Filter is not the spuriousness in this paper. We have called this method FOAF Discovery, and an example of the model is shown in Figure 2. The sequence for this method is described below.

(1) User A obtains an ID(v) which is OpenID of accessing party V, and sends the ID(v) to the RP.
(2) The user A sends a request to authorize the accessing party V to his/her OP.

4.2 Identifying cases of LOAF possession

When introducing LOAF into the system it is necessary to consider various cases of possessing or not possessing a user’s LOAF. These cases are determined by the following factors:
- Whether the accessing party presents his/her own LOAF.
- Whether the user possesses LOAFs received from each of his/her friends.

As such the FOAF Discovery can be divided into cases as shown in Table 1.

<table>
<thead>
<tr>
<th>Accessing party LOAF</th>
<th>Friend’s LOAF</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) Not brought</td>
<td>Do not have</td>
</tr>
<tr>
<td>(b) Brought</td>
<td>Do not have</td>
</tr>
<tr>
<td>(c) Not brought</td>
<td>Have</td>
</tr>
<tr>
<td>(d) Brought</td>
<td>Have</td>
</tr>
</tbody>
</table>

Four different patterns of the FOAF Discovery method are described below.

4.2.1 Pattern A

1. Send requests to the OP of all friends based on the OpenIDs in the user’s friend list.
2. Check whether any of the user’s friends are present in the LOAF received from the accessing party.

4.2.2 Pattern B

1. Send a request only to the OP of the friend included in the LOAF.
Features: Accessing party provides his/her own LOAF.

4.2.3 Pattern C
1. Check whether the accessing party's ID is included in any of the LOAFs from the user's friends that are stored on the server.
2. Send requests only to the OPs of friends whose LOAFs contain the accessing party's ID.

Features: User retains both the ID and LOAF his/her own friends on the server.

4.2.4 Pattern D
1. Check whether the ID of any friends is on the LOAF received from the accessing party.
2. If the accessing party's LOAF contains a friend, check whether the accessing party's ID is in the LOAF of that friend.
3. Send a request to the friend's OP only if the accessing party's ID is in their LOAF.

Features: Accessing party provides a LOAF. User retains both ID and LOAF of all friends on the server.

By applying the LOAF, it should be possible to reduce the number of requests that must be sent to the OPs of friends.

5. Evaluation

5.1 Evaluation of practicality

In this section, we compare the FOAF Discovery method proposed in section 4 with the case of not using the LOAF and evaluate how much requests to the OPs of friends are reduced.

5.1.1 Appropriate LOAF size for operation

As of July, 2008, mixi, the largest SNS site in Japan, had 15 million users, and the average number of registered friends was 21. This is quite large for a community site, and should be adequate for considering human relationships in a networked society. In this section we will evaluate the FOAF Discovery method using the scale of mixi as a standard.

As a characteristic of Bloom filters, the false-positive rate varies with the length of the filter and the number of elements. The false-positive rate, \( E \), for the number of hash functions, \( k \), the filter length, \( m \) (bits), and the number of registered elements, \( n \), is given by Equation (1).

\[
E = (1 - e^{-\frac{mn}{k}})^k \tag{1}
\]

The number of friends registered on mixi is limited to 1000, so we will consider the LOAF size when 1000 friends are registered. Figure 3 shows false positive rates when the number of registered friends is fixed at 1000, and the LOAF size is varied.

The optimum number of hash functions for a given the filter length and number of registered elements is given by Equation (2).

\[
k = \ln 2 \times \frac{m}{n} \tag{2}
\]

Using equations (1) and (2) above, with 1000 registered friends and setting the false positive rate at 1%, the optimum number of hash functions is six. Also, the LOAF size when using six hash functions would be about 1200 bytes.

For Patterns B and D, passing LOAFs through the network should have no significant effect on the system. However, for Patterns C and D, further consideration is required for friends' LOAF retained on the OP server. If the total LOAF size is calculated as (single LOAF size) \( \times \) (Avg. no. of friends) \( \times \) (all users), the total size of all LOAFs stored in the system would be over 378 GBytes.

5.1.2 Number of inquiries from user to friends

Using 1200 bytes as the size of the LOAF as in the previous section, we calculate the number of requests to the OPs of friends due to a single party requesting access. From a system operation perspective we can expect the access rate from spammers to be much higher than from legitimate users. Because of this, we consider only accesses from an arbitrary (single) address, and look at the number of accesses to OPs of friends, taking the LOAF false-positive rate into consideration. This is examined for each of the patterns, A to D, as divided up in section 4.2.

Table 2. Number of inquiries to OP of friend

<table>
<thead>
<tr>
<th>Number of friends</th>
<th>Number of inquiries</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Number of friends</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>500</td>
<td>500</td>
</tr>
<tr>
<td>1000</td>
<td>1000</td>
</tr>
</tbody>
</table>

Table 2 shows the number of requests in each case for different numbers of registered users. Requests are sent to all friends in Pattern A, which does not use the LOAF, and for numbers of registered friends up to 500, almost no inquiries are sent for Patterns B, C and D.

5.2 Evaluation of safety

In this section, we summarize the types of threat arising from use of the LOAF, and examine vulnerability to each threat for each of the usage patterns.

5.2.1 Ex-girlfriend attack

If a LOAF can be obtained, known IDs can be tested to find out whether they are in the LOAF or not.

Pattern B: When the user attempts access, the trustworthiness of the accessed site cannot be guaranteed, so there is a possibility that the user's LOAF will be passed to a malicious user, making an attack easy.

Pattern C: Only LOAFs from friends are used, and they are passed between trustworthy systems, resulting in less vulnerability to attack.

Pattern D: When the user attempts access, the trustworthiness of the accessed site cannot be guaranteed, so there is a possibility that the LOAF is passed to a malicious user, making an attack easy.
5.2.2 Marc Canter Attack

If all of the bits of a Bloom filter are set to one, all IDs will pass through the filter.

**Pattern B:** A LOAF supplied by the accessing party is used, so by providing a LOAF with a high density of 1 bits, he/she can impersonate a friend of a friend, making attacks easy. A check for high-density filters is necessary.

**Pattern C:** Only LOAFs of friends are used, and they are passed between trustworthy systems, resulting in less vulnerability to attack.

**Pattern D:** A LOAF provided by the accessing party is used, but the filter is applied again using the friend's LOAF, minimizing vulnerability to attack.

5.2.3 Dictionary attack

A LOAF can be falsified by registering OpenIDs created using a dictionary.

**Pattern B:** Since a LOAF submitted by the user himself/herself is used, a malicious user could create a test LOAF and submit it.

**Pattern C:** Only LOAFs of friends are used, and they are passed between trustworthy systems, resulting in less vulnerability to attack.

**Pattern D:** A LOAF provided by the accessing party is used, but the filter is applied again using the friend's LOAF, minimizing vulnerability to attack.

5.2.4 ME TOO ATTACK

An accessing party could impersonate another person by presenting their LOAF as their own.

**Pattern B:** Since the accessing party submits their own LOAF, this is vulnerable to attack. Checking that the LOAF belongs to the accessing party is necessary.

**Pattern C:** Only LOAFs of friends are used, and they are passed between trustworthy systems, resulting in less vulnerability to attack.

**Pattern D:** A LOAF provided by the accessing party is used, but the filter is applied again using the friend's LOAF, minimizing vulnerability to attack.

5.2.5 Vulnerability to attack in each case

Vulnerability to each of these types of attack in each case is shown in Table 3.

<table>
<thead>
<tr>
<th>Threat</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ex-Girlfriend attack</td>
<td>×</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>Marc Canter attack</td>
<td>×</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>Dictionary attack</td>
<td>×</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>Me Too attack</td>
<td>×</td>
<td>×</td>
<td></td>
</tr>
</tbody>
</table>

○:Not vulnerable to attack ×:Vulnerable to attack

5.3 Evaluation and discussion

Evaluation of each of the four patterns for FOAF Discovery is discussed below in terms of practicality and safety.

5.3.1 Evaluation of practicality and discussion

By using the LOAF, it is possible to reduce the number of requests to the OPs of friends dramatically, but when examining individual patterns B, C, and D, no significant difference was seen, and all were effectively reduced to zero.

With the maximum number of registered friends set to 1000, the size of a single LOAF was set to 1200 bytes. This is small enough not to create a significant load when passing through networks, but 378 GBytes are required to store all friends' LOAFs.

Also, when using friends' LOAFs, other issues must be considered, such as updating the LOAFs periodically, so from a practical standpoint, Pattern B may be the most appropriate.

5.3.2 Evaluation of safety and discussion

As shown in Table 3, Patterns B and D are vulnerable to some of the attacks described above, because they involve accepting LOAFs from, or passing LOAFs to untrusted users. Because of this, Pattern C may be the most appropriate.

6. Conclusion

In this paper, we have proposed a method called FOAF Discovery, which uses the LOAF for implementation of friend-of-a-friend access control on systems that use OpenID as their user ID. We also studied the FOAF Discovery method from the perspectives of practicality and safety, for various cases where the LOAF was used or not used.

From the practicality perspective, we showed that the number of requests to the OPs of friends could be reduced by using the LOAF. We also computed the required size of the LOAF and showed that it could be small enough not to affect systems significantly when being exchanged over networks.

From the safety perspective, we described the types of threats that could occur due to use of the LOAF, and examined vulnerability to these threats for various patterns of how the LOAF could be used.

Issues requiring further study include finding a ways to update LOAFs efficiently when friends' LOAFs are being stored on an OP server, and ways to ensure that malicious users do not falsify their LOAF.

7. REFERENCES