VOOT specifications

Introduction

VOOT 0.9 is a simple protocol for cross-domain read-only access to information about users' group membership within an organization or aggregated across organizations and their role in these groups. It can be seen as making a subset of LDAP-like information available through a web service.

The API is loosely based on the OpenSocial specification and a first iteration during GN3-JRA3-T2 project. This is just for historical reasons and not all requirements of OpenSocial nor previous versions of VOOT are met. One such example is that only the JSON data format is supported. This specification does however aim to be backwards compatible requiring little to know change to existing VOOT software.

Notational Conventions

The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and 'OPTIONAL' in this document are to be interpreted as described in [RFC 2119].

Use Cases

All the use cases that are valid for accessing group membership data from an organization's LDAP are also valid for VOOT. For instance, requesting information about users' group memberships and the members of a group. However, as opposed to LDAP, VOOT is not meant to perform user authentication.

Web applications sometimes have the ability to interface with LDAP directly to access information about users in the organization's directory (and authenticate users), but this is notoriously hard or even impossible to get working cross-domain in a secure way, except by using RADIUS, but this is not supported by web browsers, so out of scope.

Provider

This document describes the protocol used between application (client) and VOOT provider. This documents specifies the VOOT API, which is a REST API. The protocol is protected using an authentication mechanism described in the next section.

Authentication

An application (client) has two ways to authenticate to the VOOT provider, it is up to the provider to support one, or both, of those:

- Using Basic Authentication [RFC 2617] if the VOOT provider fully trusts the client as the client will have full access to all data at the provider;
• OAuth 2.0 [RFC 6749] if there is some trust between the VOOT provider and
the client where it is left to the user to explicitly authorize the client
that wants to access data from the provider. The user gives the client only
access to their data, and not to that of other users at the VOOT provider.

Technically, OAuth 2.0 is an authorization framework and not intendend for
authentication, however, from the point of view of the VOOT provider the
client authenticates to the API using an access token.

API

The VOOT API supports two calls:

• Retrieve a list of groups the user is a member of;
• Retrieve the list of people that are member of a group the user is also a
  member of.

For the API calls one has to specify the user identifier for which the
information is retrieved. For API calls using OAuth 2.0 the @me placeholder
identifier MUST be supported which is replaced in the provider with the actual
user identifier that authorized the client to act on its behalf. With this
placeholder the client accessing information about the user does not need to
know the unique identifier of the user at the provider. Specifying an user
identifier MAY be supported by the OAuth 2.0 protected provider, however this
is not recommended.

For Basic Authentication an actual user identifier and group identifier MUST be
specified, @me is not supported here due to the lack of binding with an user
that authorized the client to act on its behalf. It is out of scope how the
client obtains the identifiers of the user, or group.

If the user john authorized a client to act on its behalf, with an OAuth 2.0
protected provider the following calls are defined, of which only the first one
MUST be implemented:

/groups/@me
/people/@me/students

If information about the user john is queried by a client using Basic
Authentication the following calls are defined, of which only the first one
MUST be implemented:

/groups/john
/people/john/students

OAuth

In order to retrieve data from the VOOT service, it is required that the client
specifies what type of call it wants to perform. For any of the calls a
different scope is required. Multiple scopes can be combined:

• Retrieving a list of groups the user is a member of, requires the scope
  http://openvoot.org/groups
• Retrieving the list of people that are member of a group the user is also a
  member of, also requires the scope http://openvoot.org/people

For backwards compatibility the scope read will give access to all the
above, assuming there are no other server side restrictions.

Retrieve Group Membership

This call retrieves a list of groups the user is a member of:

/groups/@me

or
Where {{userId}} is replaced with an identifier of the user at the provider. This call MUST be supported. The response can include the following keys:

- (REQUIRED) id: The, to the provider, local unique identifier of the group;
- (REQUIRED) voot_membership_role: The role the user has in this group;
- (OPTIONAL) title: The short human readable name of the group;
- (OPTIONAL) description: A description of the group.

The id field SHOULD be opaque to the client. The field voot_membership_role can be any of these values: admin, manager or member.

Retrieve Members of a Group

This call retrieves a list of all members of a group the user is a member of:

```
/people/@me/{{groupId}}
```

or

```
/people/{{userId}}/{{groupId}}
```

Where {{userId}} is replaced with an identifier of the user at the provider and {{groupId}} is replaced with a group identifier which was obtained through for instance the call used to retrieve group membership for a particular user.

This call MAY be supported. The response can include the following keys:

- (REQUIRED) id: The, to the provider, local unique identifier of the user;
- (OPTIONAL) displayName: The name by which the user prefers to be addressed;
- (OPTIONAL) voot_membership_role: The role the user has in this group;
- (OPTIONAL) emails: The email address(es) of the user;

The id field SHOULD be opaque to the client. The field voot_membership_role can be any of these values: admin, manager or member. The user MUST be a member, but not necessary have the role member of the group being queried. The emails field SHOULD contain a list of email addresses which provides the type to be any of work, home or other. For example:

```
"emails": [
  {
    "type": "work",
    "value": "bmcatee@students.example.edu"
  }
]
```

Request Parameters

The API calls have three OPTIONAL parameters that manipulate the result obtained from the provider:

- sortBy
- startIndex
The `sortBy` parameter determines the key in the result that is used for sorting the groups or group members. It can be any of the mentioned keys mentioned in the previous section:

- `id`
- `displayName`
- `title`
- `description`
- `voot_membership_role`

If the key is not available in the set being sorted, the results are not sorted or sorted at the provider’s discretion. It is up to the provider whether or not to sort and by what key in what order if these parameters are not present.

If the results are to be sorted, the value SHOULD be compared as strings and SHOULD be sorted case-insensitive in ascending order.

The `startIndex` parameter determines the offset as an integer >= 0 at which the start for giving back results. The `count` parameter, an integer >= 0 indicates the number of results to be given back. The `startIndex` and `count` parameters can be used to implement paging by returning only a subset of the results. These parameters are OPTIONAL, if they are not provided or invalid the provider MUST consider `startIndex` equal to 0 and `count` equal to the total number of items available in the entire set for the particular query.

The sorting, if requested, MUST be performed on the provider before applying limiting the results based on the `startIndex` and `count` parameters.

For the API call, requesting user information, the `sortBy` parameter has no effect. Using `startIndex` and `count` is possible, however they are of little use as there always will be only one answer, assuming the user exists.

**Response Parameters**

All responses mentioned above have the same JSON structure. There are always four keys:

- `startIndex`
- `itemsPerPage`
- `totalResults`
- `entry`

Where `startIndex` contains the offset from which the results are returned, this is usually equals to the requested `startIndex`. The `itemsPerPage` contains the actual number of results in the response set, as part of `entry`, returned. The `totalResults` field contains the full number of elements available, not depending on the `startIndex` and `count` parameters.

The `entry` key contains a list of items, either groups, people or person information. Below are some examples.

**API Examples**

Below are some API examples for retrieve group membership, a list of group members and information about the user.

**Retrieve Group Membership**

This is an example of the response to the query:

```plaintext
Host: provider.example.org
GET /groups/@me?sortBy=title HTTP/1.1
```

The response looks like this:

```json
HTTP/1.1 200 OK
Content-Type: application/json

{
    "entry": [],
    "description": "Group containing employees."
}
```
Retrieve Members of a Group

This is an example of the response to the query:

```
Host: provider.example.org
GET /people/@me/members?sortBy=displayName&startIndex=3&count=2 HTTP/1.1
```

The response looks like this:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "entry": [
    {
      "displayName": "Bobby Mcatee",
      "emails": [
        {
          "type": "work",
          "value": "bmcatee@students.example.edu"
        }
      ],
      "id": "bmcatee",
      "voot_membership_role": "member"
    },
    {
      "displayName": "Myra Wisdom",
      "emails": [
        {
          "type": "home",
          "value": "mwisdom@students.example.edu"
        }
      ],
      "id": "mwisdom",
      "voot_membership_role": "member"
    }
  ],
  "itemsPerPage": 2,
  "startIndex": "3",
  "totalResults": "7"
}
```

Error Handling

Handling failures of Authentication, either Basic or Bearer are handled in the ways described in [RFC 2617] and [RFC 6750]. This will involve sending the WWW-Authenticate header if something is wrong, for example an invalid OAuth 2.0 access token will result in the following response:
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Bearer realm="Resource Server",error="invalid_token",error_description="the access token is not valid"
Content-Type: application/json

{"error":"invalid_token","error_description":"the access token is not valid"}

There are also some request errors defined, i.e.: invalid requests to the provider that should be dealt with in a certain manner. Only the call that retrieves group membership MUST be supported, the other calls do not need to be supported. When this call is disabled a response code of 400 Bad Request is returned with error set to invalid_request.

The error response is returned as JSON, for example:

HTTP/1.1 404 Not Found
Content-Type: application/json

{
  "error": "invalid_user",
}

The error field MUST be present.

Retrieve Group Membership

The call looks like this:

/groups/@me

- If Basic Authentication is used and @me is used an error response with code 404 Not Found is returned. The error field contains invalid_user.
- If a user identifier is specified instead of @me for providers not supporting the use of user identifiers the same error is returned;
- If the specified user does not exist at the provider an error response with code 404 Not Found is returned. The error field contains invalid_user;
- If any other error occurs an error response with code 500 Internal Server Error is returned. The error field contains internal_server_error.

Retrieve Members of a Group

The call looks like this:

/people/@me/members

- If Basic Authentication is used and @me is used an error response with code 404 Not Found is returned. The error field contains invalid_user.
- If a user identifier is specified instead of @me for providers not supporting the use of user identifiers the same error is returned;
- If the specified user does not exist at the provider an error response with code 404 Not Found is returned. The error field contains invalid_user;
- If the specified user is not a member of the group an error response with code 403 Forbidden is returned. The error field contains not_a_member. This response MUST be returned when the user is not a member, no matter whether the group actually exists or not;
- If any other error occurs an error response with code 500 Internal Server Error is returned. The error field contains internal_server_error.

Proxy Operation

One of the use cases is to make it possible to combine data from various group providers using one API service. This way group membership information
can be aggregated from various sources. The proxy provides an OAuth 2.0 protected API to clients and in the backend uses Basic Authentication to talk to the group providers from which it needs to aggregate data.

```
+-------+              +----------+
  |       |              | VOOT     |
  |       +--------------+ Provider |
  |       |  VOOT/Basic  | A        |
  | VOOT  |              +----------+
--------------+ Proxy |
VOOT/OAuth  |       |              +----------+
  |       |  VOOT/Basic  | B        |
  |       +--------------+ Provider |
+-------+              +----------+
```

From the client's point of view there should be no difference in the API compared to talking directly to a group provider. There are however some special error cases that should be considered. For instance if (one of) the remote group providers is not available. Also the group identifiers that were scoped locally per group provider need to be modified to include a 'scope', i.e. to indicate to what group provider they belong.

For example the user john, which is a local identifier at a group provider can occur in multiple group providers, so it needs to be matched to this particular group provider, for example prefixed with the identifier of the group provider. The prefixed value SHOULD be opaque to the client as well.

Identity Federation Considerations

The VOOT protocol is not meant to authenticate users at a service. For this, other mechanisms such as federated identity protocols like SAML, OpenID Connect or Mozilla Persona exists. What needs to be considered here is making sure the identifiers used in the authentication layer are the same as used by the group provider when exposing this information to clients.

In SAML for instance there is the persistent opaque identifier provided to Service Provider (SP) through the NameID value. This is an identifier that remains constant for a particular user per SP. This way, SPs cannot match the same user between themselves based on this identifier alone.

This only needs to be considered when implementing the call that retrieves group members. For just retrieving the group identifiers there can be any mechanism to make the values opaque as this information is typically not provided through the SAML assertion.

Roles

The roles admin, manager and member are not well defined. Their actual semantics is based on what the application considers to be an admin, manager or member. However, in general, the following can be said about the roles:

- A user with role admin has more rights than a user with role manager or role member;
- A user with role manager has more rights than a user with role member;

For more fine grained roles the use of an entitlement based system is recommended, which is out of scope of this specification.

Privacy

In order to maintain user privacy only the group membership API call should be allowed by third party clients. The other calls are not needed to determine group membership, e.g. to base application authorization on. If a user is a member of a particular group certain privileges may be granted based on just this fact.

Only the @me user identifier should be allowed as to avoid needing to obtain unique user identifiers.
If one makes use of the proxy scenario where the proxy provider is trusted, Basic Authentication can be used with for instance the local \textit{uid} of the user. The proxy then \textbf{SHOULD} take care of making this information opaque towards the client and generate new identifiers for the same user for different clients.

The calls to retrieve group members are unacceptable from the point of view of user privacy. Even if OAuth 2.0 is used this would leak user identifiers of other people to clients without the users that are a member of this group ever gave permissions for that. The impact of this can be minimized by making sure only the unique, opaque user identifier of the user is mentioned in the result and not other identifying information such as \textit{displayName}. However, without these additional attributes this API call will be less useful.

References

- RFC 2119 Key words for use in RFCs to Indicate Requirement Levels
- RFC 2617 HTTP Authentication: Basic and Digest Access Authentication
- RFC 6749 The OAuth 2.0 Authorization Framework
- RFC 6750 The OAuth 2.0 Authorization Framework: Bearer Token Usage

Thanks

- Maarten Kremers <maarten.kremers AT surfnet.nl>
- Andreas Åkre Solberg (||)
- Leif Johansson (||)

Author

- François Kooman <francois.kooman@surfnet.nl>