XEP-0356: Privileged Entity

Jérôme Poisson

mailto:goffi@goffi.org
xmpp:goffi@jabber.fr

2023-04-24
Version 0.4.1

Status	Type	Short Name
Experimental	Standards Track	NOT_YET_ASSIGNED

This specification provides a way for XMPP entities to have a privileged access to some other entities data

Legal

Copyright

This XMPP Extension Protocol is copyright © 1999 – 2024 by the XMPP Standards Foundation (XSF).

Permissions

Permission is hereby granted, free of charge, to any person obtaining a copy of this specification (the "Specification"), to make use of the Specification without restriction, including without limitation the rights to implement the Specification in a software program, deploy the Specification in a network service, and copy, modify, merge, publish, translate, distribute, sublicense, or sell copies of the Specification, and to permit persons to whom the Specification is furnished to do so, subject to the condition that the foregoing copyright notice and this permission notice shall be included in all copies or substantial portions of the Specification. Unless separate permission is granted, modified works that are redistributed shall not contain misleading information regarding the authors, title, number, or publisher of the Specification, and shall not claim endorsement of the modified works by the authors, any organization or project to which the authors belong, or the XMPP Standards Foundation.

Warranty

NOTE WELL: This Specification is provided on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE.

Liability

In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall the XMPP Standards Foundation or any author of this Specification be liable for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising from, out of, or in connection with the Specification or the implementation, deployment, or other use of the Specification (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if the XMPP Standards Foundation or such author has been advised of the possibility of such damages.

Conformance

This XMPP Extension Protocol has been contributed in full conformance with the XSF's Intellectual Property Rights Policy (a copy of which can be found at <https://xmpp.org/about/xsf/ipr-policy> or obtained by writing to XMPP Standards Foundation, P.O. Box 787, Parker, CO 80134 USA).

Contents

1 Introduction

XMPP components are used for long through Jabber Component Protocol (XEP-0114) [1], but are quite limited: they have a restricted access to other entities data, similar to what a client can do. This is sufficient for components like gateways, but very limiting for more complex components like a PubSub service. The goal of this XEP is to allow a component or any entity to have a "privileged" status, and access some other entity data with the same privileges than the entity itself, that means manage an entity roster on its behalf, send <message/>, <iq/> or receive <presence/> stanzas in the name of the server.

Privileged entities have numerous advantages, including:

- a step forward in decentralization: it is possible for an entity to do tasks which were before reserved to server itself. For example, a privileged pubsub component can offer access model based on publisher's roster

- better integration of components: a gateway can add items to an entity roster itself

- possibility to overpass a server limitation (typically: incomplete Personal Eventing Protocol (XEP-0163) [2] implementation)

- quick development cycle: developers can implement the components they need without waiting for a new server release

- server agnostic

Privileged entity has been created with the main goal to create an external, server agnostic, PEP service. It is restricted to only a couple of features, see Acknowledgements section for more details.

This XEP is complementary to Namespace Delegation (XEP-0355) [3] (and works in a similar way), although they can be used together or separately. To build something like an external PEP service, it is necessary to use both XEPs.

2 Requirements

A privileged entity must be able to do what a PEP service can do and to access roster, so it must be able to (according to configuration):

- get and modify the roster of any entity managed by the server, and optionally get roster pushes

---

[1]XEP-0114: Jabber Component Protocol <https://xmpp.org/extensions/xep-0114.html>.

[2]XEP-0163: Personal Eventing Protocol <https://xmpp.org/extensions/xep-0163.html>.

[3]XEP-0355: Namespace Delegation <https://xmpp.org/extensions/xep-0355.html>.

- send a <message/> stanza on behalf of the server

- send a <iq/> stanza on behalf of a server user

- access <presence/> informations for entities in a managed entity's roster (and for managed entity itself)

The privilege mechanism MUST be totally transparent for the managed entities.

3 Glossary

- **Privileged entity** — the entity which has a privileged status.

- **Managed entity** — the entity that is managed by a privileged entity.

4 Accessing Roster

4.1 Server Allows Roster Access

Roster access is granted in the server configuration. Roster access can have 4 types:

- **none** — the entity is not allowed to access managed entity roster at all. This is usually the default value.

- **get** — the entity is allowed to send <iq/> stanzas of type *'get'* for the namespace 'jabber:iq:roster'.

- **set** — the entity is allowed to send <iq/> stanzas of type *'set'* for namespace 'jabber:iq:roster'.

- **both** — the entity is allowed to send <iq/> stanzas of type *'get'* and *'set'* for namespace 'jabber:iq:roster'.

Roster access MAY have an optional "push" argument which can be set to "true" or "false" and SHOULD default to "true" if roster permission access type is either "get" or "both", and to "false" otherwise.
If set to "true", the server will send roster pushes as explained below.. If set to "false", the server MUST NOT send roster pushes. The "false" value is mostly there to save resources if the managing entity knows that it doesn't need to be notified of roster updates. Roster pushes MUST NOT be sent if roster permission type is either "none" or "set".

4.2   Server Advertises Entity Of Allowed Permission

Once a privileged entity is authenticated and stream is started, the server send it a <message/> stanza with a <privilege/> elements which MUST have the namespace 'urn:xmpp:privilege:2'. This element contains <perm/> elements which MUST contain a 'access' attribute of the value "roster" and a 'type' attribute which must correspond to the type configured as specified in "Server Allows Roster Access" section.
The <perm> element MAY contain a 'push' attribute with a value of either "true" or "false" according to configuration. If the 'push' attribute is omitted, it defaults to "true" if "roster" permission is "get" or "both", otherwise it's set to "false". If 'push' is "true", roster pushes MUST be transmitted, if 'push' is "false" they MUST NOT be transmitted.

Listing 1: Server Advertises Roster Privilege

```
<message from='capulet.net' to='pubub.capulet.lit' id='12345'>
  <privilege xmlns='urn:xmpp:privilege:2'>
    <perm access='roster' type='both' push='true'/>
  </privilege>
</message>
```

Here *pubsub.capulet.lit* is allowed to do *get* and *set* operations on all entities managed by capulet.lit

4.3   Privileged Entity Manage Roster

Doing a *get* or *set* operation on the roster of a managed entity is done in the usual way (as described in RFC 6121 [4] section 2), except that the 'to' attribute is set to the attribute of the managed entity. The server MUST check that the privileged entity has right to *get* or *set* the roster of managed entity, and MUST return a <forbidden/> error if it is not the case:

Listing 2: Privileged Entity Get Managed Entity Roster

```
<iq id='roster1'
    from='pubsub.capulet.lit'
    to='juliet@example.com'
    type='get'
    id='roster1'>
      <query xmlns='jabber:iq:roster'/>
</iq>
```

The server then answers normally, as it would have done to the managed entity:

---

[4]RFC 6121: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence <http://tools.ietf.org/html/rfc6121>.

Listing 3: Server Answers To Privileged Entity

```
<iq id='roster1'
  from='juliet@example.com'
  to='pubsub.capulet.net'
  type='result'>
  <query xmlns='jabber:iq:roster' ver='ver7'>
    <item jid='nurse@example.com'/>
    <item jid='romeo@example.net'/>
  </query>
</iq>
```

4.4 Server Sends Roster Pushes

If "push" attribute is unset or set to "true" and roster permission type is either "get" or "both", the server MUST send roster pushes when there is a newly created, updated or deleted roster item for roster of any managed entity. A roster push is built as explained in XMPP IM [5] with a 'from' attribute explicitly set to the bare jid of the managed entity.

Listing 4: Server Send Roster Push of Juliet to Privileged Entity

```
<iq id='roster_push_1'
  from='juliet@capulet.lit'
  to='pubsub.capulet.lit'
  type='set'>
  <query xmlns='jabber:iq:roster'>
    <item jid='nurse@example.com'/>
  </query>
</iq>
```

5 Message Permission

5.1 Authorizing Messages

In some cases, it can be desirable to send notifications (e.g. PEP service), so the privileged entity must be able to send <message/> stanzas. This is allowed in server configuration in the same way as for roster permission. The permission type can have the following values:

- **none** — the entity is not allowed to send <message/> stanza in the name of the server. This is usually the default value.

- **outgoing** — the entity is allowed to send <message/> stanzas in the name of the server, according to following restrictions.

---

[5]RFC 6121: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence <http://tools.ietf.org/html/rfc6121>.

A privileged entity can then send message on the behalf either of the server or of a bare JID of an entity managed by the server (i.e. a bare jid with the same domain as the server), using Stanza Forwarding (XEP-0297) [6]. The <forwarded/> element MUST be a child of a <privilege/> element with a namespace of *'urn:xmpp:privilege:2'*, with the following restriction:

1. forwarded <message/> 'from' attribute MUST be a bare JID from the server, no resource is allowed

If this rule is violated, the server MUST return a <message/> error with condition <forbidden/>, as in RFC 6120 [7] section 8.3.3.4.

5.2 Advertising Permission

Server advertises "message" permission in the same way as for "roster" permission, except that 'access' attribute has the value of "message", and the 'type' attribute as a value of 'outgoing':

Listing 5: Server Advertises Roster And Message Privileges

```
<message from='capulet.net' to='pubub.capulet.lit' id='54321'>
    <privilege xmlns='urn:xmpp:privilege:2'>
        <perm access='roster' type='both' push='true'/>
        <perm access='message' type='outgoing'/>
    </privilege>
</message>
```

5.3 Sending Messages

Now that *pubsub.capulet.lit* is allowed, it can send messages using <forwarded/> elements.

Listing 6: privileged entity send a notification message

```
<message from='pubsub.capulet.lit' to='capulet.lit' id='notif1'>
  <privilege xmlns='urn:xmpp:privilege:2'>
    <forwarded xmlns='urn:xmpp:forward:0'>
      <message from='juliet@capulet.lit'
         id='foo'
         to='romeo@montague.lit/orchard'
         xmlns='jabber:client'>
         <event xmlns='http://jabber.org/protocol/pubsub#event'>
           <items node='http://jabber.org/protocol/tune'>
             <item>
```

---

[6]XEP-0297: Stanza Forwarding <https://xmpp.org/extensions/xep-0297.html>.
[7]RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.

```
            <tune xmlns='http://jabber.org/protocol/tune'>
              <artist>Gerald Finzi</artist>
              <length>255</length>
              <source>Music for "Love's␣Labors␣Lost" (Suite for
                  small orchestra)</source>
              <title>Introduction (Allegro vigoroso)</title>
              <track>1</track>
            </tune>
          </item>
        </items>
      </event>
      <delay xmlns='urn:xmpp:delay' stamp='2014-11-25T14:34:32Z'/>
    </message>
  </forwarded>
  </privilege>
</message>
```

The server sees that forwarded message 'from' attribute (*juliet@capulet.lit*) is a bare JID of the server, and that outgoing message permission was granted; it can now send the notification:

Listing 7: server sends the notification as if it was originating from him

```
<message from='juliet@capulet.lit'
  id='bar'
  to='romeo@montague.lit/orchard'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <items node='http://jabber.org/protocol/tune'>
      <item>
        <tune xmlns='http://jabber.org/protocol/tune'>
          <artist>Gerald Finzi</artist>
          <length>255</length>
          <source>Music for "Love's␣Labors␣Lost" (Suite for small
              orchestra)</source>
          <title>Introduction (Allegro vigoroso)</title>
          <track>1</track>
        </tune>
      </item>
    </items>
  </event>
  <delay xmlns='urn:xmpp:delay' stamp='2014-11-25T14:34:32Z'/>
</message>
```

6   IQ permission

6.1   IQ Stanzas on Behalf of a Server User

It may be necessary for a component to send <iq/> stanzas on behalf of a server user. This is, for instance, the case for a pubsub component willing to implement Pubsub Account

Management (XEP-0376) [8].

To do this, an "iq" permission must be granted in server configuration.

To grant an "iq" permission, authorised namespaces must be specified and associated to a value indicating the type of <iq/> request which are allowed. The value is similar to the roster access type, there are 4 values possible:

- **none** — the entity is not allowed to send <iq/> stanzas for this namespace.

- **get** — the entity is allowed to send <iq/> stanzas of type *'get'* for this namespace.

- **set** — the entity is allowed to send <iq/> stanzas of type *'set'* for this namespace.

- **both** — the entity is allowed to send <iq/> stanzas of type *'get'* and *'set'* for this namespace.

6.2 Advertising Permission

Server adversites "iq" permission by using a <perm> element with an 'access' attribute of the value "iq" and wihout 'type' attribute. This element MAY contain any number of <namespace> elements with a 'ns' attribute of the value of the granted namespace, and a 'type' attribute with one of the value indicated in previous section

Listing 8: Server Advertises IQ Privilege

```
<message from='capulet.net' to='pubub.capulet.lit' id='iq_perm_1'>
  <privilege xmlns='urn:xmpp:privilege:2'>
    <perm access='iq'>
      <namespace ns='http://jabber.org/protocol/pubsub' type='set' />
    </perm>
  </privilege>
</message>
```

Here *pubsub.capulet.lit* is allowed to send <iq/> stanza of type *set* with the namespace 'http://jabber.org/protocol/pubsub' on behalf of any entity managed by capulet.lit

6.3 Sending IQ Stanzas

Sending an <iq/> stanza on behalf of a user is done by following those steps:

- generate the <iq/> stanza which much be sent on behalf of the user, we call it "encapsulated <iq/> stanza". The 'type' attribute and the namespace of the payload element must match the "iq" permission granted by the server. This <iq/> stanza MUST have a namespace of "jabber:client"

---

[8]XEP-0376: Pubsub Account Management <https://xmpp.org/extensions/xep-0376.html>.

- the encapsulated <iq/> stanza MUST either have no 'from' attribute, or a 'from' attribute set to the bare jid of the entity on behalf of who the privileged entity is doing the request

- encapsulate the <iq/> request in a <privileged_iq> element with a namespace of 'urn:xmpp:privilege:2'

- use the <privileged_iq> element as payload of a top-level <iq/> request adressed to the bare JID of the managed entity

- use the same type for top-level <iq/> request as for the encapsulated <iq/> request

The server MUST refuse the request with a <forbidden/> error if any of the following condition happens:

- the 'to' attribute of the top-level <iq/> stanza is not a bare JID of a managed entity

- the requesting entity has not the permission granted for the namespace used in payload of the encapsulated <iq/> stanza

- the requesting entity has not the permission granted for the type of <iq/> request used in the encapsulated <iq/> stanza for the namespace used in its payload

- the namespace of the encapsulated <iq/> stanza is not "jabber:client"

- the 'from' attribute of the encapsulated <iq/> stanza exists and is set to a JID which doesn't match the 'to' attribute of the top-level <iq/> stanza.

- the 'type' attribute of the top-level <iq/> stanza does not match the 'type' attribute of the encapsulated <iq/> stanza

Once the server gets the <iq/> response, it sends it back to privileged entity using a Stanza Forwarding (XEP-0297) [9] <forward> element encapsulated in a <privilege> element with a namespace of 'urn:xmpp'.

In following example, the pubsub component *pubsub.capulet.lit* handles Pubsub Account Management (XEP-0376) [10]. After getting a pubsub subscribe request, from Juliet to subscribe to Romeo's blog, it forward it to Romeo's server:

Listing 9: Privileged Entity Send an <iq/> Stanza on Behalf of Juliet

```
<iq
        from='pubsub.capulet.lit'
        to='juliet@capulet.lit'
        type='set'
        id='priv_iq_1'>
    <privileged_iq xmlns='urn:xmpp:privilege:2'>
```

---

[9]XEP-0297: Stanza Forwarding <https://xmpp.org/extensions/xep-0297.html>.
[10]XEP-0376: Pubsub Account Management <https://xmpp.org/extensions/xep-0376.html>.

```
        <iq
          xmlns='jabber:client'
          type='set'
          to='romeo@montaigu.lit'
          id='sub_1'>
          <pubsub xmlns='http://jabber.org/protocol/pubsub'>
            <subscribe
              node='urn:xmpp:microblog:0'
              jid='juliet@capulet.lit'/>
          </pubsub>
        </iq>
      </privileged_iq>
  </iq>
```

When receiving this stanza, the server does a couple of things:

- It checks that top-level <iq/> stanza is addressed to the bare JID of a managed entity: it's the case for *juliet@capulet.lit.*

- It decapsulate the encapsulated <iq/> stanza, check that it's namespace is 'jabber:client' and that its 'type' attribute has the same value as the top-level <iq/> stanza. It's "set" in both case, so it's good.

- It gets the payload of the encapsulated <iq/> stanza, and checks that *pubsub.capulet.lit* is authorised to send privilied <iq/> for its namespace with the given <iq/> type. Here the payload has a namespace of 'http://jabber.org/protocol/pubsub' and the <iq/> a type of "set", this combination is authorised for *pubsub.capulet.lit*, it's good.

- It checks that the encapsulated <iq/> stanza either has no 'from' attribute or has a 'from' attribute mathing the bare JID set in 'to' attribute of the top-level <iq/> stanza. Here no 'from' attribute is set, it's good.

- It sets the 'from' attribute of the encapsulated <iq/> stanza to same value as the 'to' attribute of the top-level <iq/> stanza (i.e. the bare JID of the managed entity).

- Once everything is checked, it can then send the encapsulated <iq/> as if it were sent by Juliet herself (the only difference is that the 'from' attribute has no resource while it would have the resource of Juliet's client if she was sending it herself).

Listing 10: Server Send the Encapsulated <iq/> Stanza

```
<iq
  xmlns='jabber:client'
  type='set'
  from='juliet@capulet.lit'
  to='romeo@montaigu.lit'
  id='sub_1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
```

9

```
      <subscribe
        node='urn:xmpp:microblog:0'
        jid='juliet@capulet.lit'/>
  </pubsub>
</iq>
```

The server will then get the response with a type of either "result" or "error" as specified in XMPP Core [11]. It sends it back to *pubsub.capulet.lit* using a Stanza Forwarding (XEP-0297) [12] <forward> element:

Listing 11: Server Gets the <iq/> Response and Forward it to Privileged Entity

```
<iq type='result'
    from='romeo@montaigu.lit'
    to='juliet@capulet.lit'
    id='sub_1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <subscription
        node='urn:xmpp:microblog:0'
        jid='juliet@capulet.lit'
        subid='some_id'
        subscription='subscribed'/>
  </pubsub>
</iq>

<iq
  type='result'
  from='juliet@capulet.lit'
  to='pubsub.capulet.lit'
  id='priv_iq_1'>
  <privilege xmlns="urn:xmpp:privilege:2">
    <forwarded xmlns='urn:xmpp:forward:0'>
      <iq type='result'
        from='romeo@montaigu.lit'
        to='juliet@capulet.lit'
        id='sub_1'>
        <pubsub xmlns='http://jabber.org/protocol/pubsub'>
          <subscription
            node='urn:xmpp:microblog:0'
            jid='juliet@capulet.lit'
            subid='some_id'
            subscription='subscribed'/>
        </pubsub>
      </iq>
    </forwarded>
  </privilege>
```

---

[11]RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.

[12]XEP-0297: Stanza Forwarding <https://xmpp.org/extensions/xep-0297.html>.

```
</iq>
```

pubsub.capulet.lit has now subscribed to Romeo's blog on behalf of Juliet.

7   Presence Permission

7.1   Managed Entity Presence

It can be often desirable for a privileged entity to have presence information of the managed entities (e.g. to know when to send them notificiations). As privileges must be transparent for the managed entity, this presence has to be sent by the server without modifying managed entity roster.

This is allowed in server configuration in the same way as for *roster* and *message* permissions. The "presence" type can have the following values:

- **none** — the entity is not allowed to access <presence/> informations at all. This is usually the default value.

- **managed_entity** — the entity is allowed to receive managed entity presence (see below).

- **roster** — the entity is allowed to receive presence informations of managed entity contacts, see Roster Presence section.

If the privilege is granted, the server MUST use a directed presence from the full jid of the managed entity, to the privileged entity, as specified in RFC 6121 [13] section 4.6, on the behalf of managed entity each time its presence information change.

Only initial <presence/> stanzas and <presence/> stanzas with a 'type' attribute with the value "unavailable" are transmitted to the privileged entity, the server MUST NOT transmit any other <presence/> stanza.

7.2   Advertising Permission

Server advertises "presence" permission in the same way as for "roster" or "message" permissions, except that 'access' attribute has the value of "presence", and the 'type' attribute has a value of "managed_entity"

7.3   Server Send presence informations

Once the "presence" permission is granted, the server send presence informations:

---

[13]RFC 6121: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence <http://tools.ietf.org/html/rfc6121>.

Listing 12: Server Receives Initial <presence/> from Juliet

```
<presence from='juliet@capulet.lit/balcony'
  id='presence1'
  xml:lang='en'>
  <show>chat</show>
  <status>Staying on the balcony</status>
</presence>
```

Listing 13: server redirects presence to privileged entity

```
<presence from='juliet@capulet.lit/balcony'
  to='pubsub.capulet.lit'
  id='presence1'
  xml:lang='en'>
  <show>chat</show>
  <status>Staying on the balcony</status>
</presence>
```

7.4 Roster Presence

In addition to "managed entity presence", a privileged entity may need to know when a contact in managed entity roster is online (for example, it's necessary for a PEP service because of the presence default access model).

As for other permissions, the access is granted in server's configuration, but there is an additional restriction: the privileged entity MUST have read permission on roster namespace (i.e. 'type' attribute in allowed <perm> of access *roster* MUST have a value of either **get** or **both**).

If the privilege is granted, the server MUST send to the privileged entity every presence information with no 'type' attribute or with a 'type' with a value of 'unavailable' that the privileged entity is receiving or would receive if it were available. It do it in the same way as for managing entity by using directed <presence/> from the full jid of the entity from which presence information has changed, to the privileged entity. If the managed entity is unavailable but the privileged entity is available, the server MUST send <presence/> stanza to the later anyway.

Having "roster" type for "presence" permission imply that you have also implicitly "managed_entity" type.

The server MUST reject the permission if the privileged entity doesn't have read permission on roster namespace.

Note: this permission should be given carefully, as it gives access to presence of potentially a lot of entities to the privileged entity (see security considerations).

7.5 Advertising Permission

Server advertises roster "presence" permission in the same way as for other permissions, except that the 'access' attribute has the value of "presence", and the 'type' attribute has a

value of "roster"

Listing 14: Server Advertises Roster, Message, Managed Entity Presence and Roster Presence
         Privileges

```
<message from='capulet.net' to='pubub.capulet.lit' id='54321'>
  <privilege xmlns='urn:xmpp:privilege:2'>
    <perm access='roster' type='both' push='true'/>
    <perm access='message' type='none'/>
    <perm access='iq'>
      <namespace ns='http://jabber.org/protocol/pubsub' type='set' />
    </perm>
    <perm access='presence' type='roster'/>
  </privilege>
</message>
```

Note the presence of *roster* permission request.

7.6 Privileged Entity Receive Roster Presences

Listing 15: server receives new presence from Romeo, which is in Juliet's roster

```
<presence from='romeo@montaigu.lit/orchard'/>
```

Listing 16: server sends the presence as usually, but also to the privileged entity

```
<presence from='romeo@montaigu.lit/orchard'
          to='juliet@capulet.lit'/>
<presence from='romeo@montaigu.lit/orchard'
          to='pubsub.capulet.lit'/>
```

8 Business Rules

1. For "presence" access, if a privileged entity is connected after first <presence/> stanzas
   have been received, the server MUST send it all the <presence/> stanzas with no 'type'
   attribute it would have had if it was connected first (in other words: all the presences
   informations for connected entities it has access to).

2. For "presence" access, if a privileged entity is supposed to received several time
   the same <presence/> stanza, the server SHOULD send it only once. For example:
   if *pubsub.capulet.lit* has a "presence" access with a "roster" type for *capulet.lit*, and
   juliet@capulet.lit and *nurse@capulet.it* both have *romeo@montague.lit* in their roster. When
   romeo is available, *pubsub.capulet.lit* shoud have its <presence/> stanza only once (in-
   stead of 2 times).

9   Security Considerations

1. Privileged entitiy has access to sensitive data, and can act as the server itself, permissions should be granted carefully, only if you absolutely trust the entity.

2. Roster presence is particulary sensitive, because presence informations of whole rosters are shared.

3. IQ permission namespaces and types must be granted carefuly, as they allow component to act on behalf of any user of the server.

4. Generaly, the server MUST NOT allow the privileged entity to do anything that the managed entity could not do.

10   IANA Considerations

This document requires no interaction with the Internet Assigned Numbers Authority (IANA) [14].

11   XMPP Registrar Considerations

11.1   Protocol Namespaces

The XMPP Registrar [15] includes 'urn:xmpp:privilege:2' in its registry of protocol namespaces (see <`https://xmpp.org/registrar/namespaces.html`>).

- urn:xmpp:privilege:2

11.2   Protocol Versioning

If the protocol defined in this specification undergoes a revision that is not fully backwards-compatible with an older version, the XMPP Registrar shall increment the protocol version number found at the end of the XML namespaces defined herein, as described in Section 4 of XEP-0053.

---

[14]The Internet Assigned Numbers Authority (IANA) is the central coordinator for the assignment of unique parameter values for Internet protocols, such as port numbers and URI schemes. For further information, see <`http://www.iana.org/`>.

[15]The XMPP Registrar maintains a list of reserved protocol namespaces as well as registries of parameters used in the context of XMPP extension protocols approved by the XMPP Standards Foundation. For further information, see <`https://xmpp.org/registrar/`>.

12 XML Schema

```
<?xml version='1.0' encoding='UTF-8'?>

<xs:schema
  xmlns:xs='http://www.w3.org/2001/XMLSchema'
  targetNamespace='urn:xmpp:privilege:2'
  xmlns='urn:xmpp:privilege:2'
  elementFormDefault='qualified'>

  <xs:element name='privilege'>
    <xs:complexType>
      <xs:element name='perm'
        maxOccurs='unbounded'>
        <xs:complexType>
          <xs:attribute name='access' use='required' type='xs:string'/
            >
          <xs:simpleType base='xs:NMTOKEN'>
            <xs:enumeration value='roster'/>
            <xs:enumeration value='message'/>
            <xs:enumeration value='presence'/>
          </xs:simpleType>
          <xs:attribute name='type' use='optional'>
            <xs:simpleType base='xs:NMTOKEN'>
              <xs:enumeration value='none'/>
              <xs:enumeration value='get'/>
              <xs:enumeration value='set'/>
              <xs:enumeration value='both'/>
              <xs:enumeration value='outgoing'/>
              <xs:enumeration value='managed_entity'/>
              <xs:enumeration value='roster'/>
            </xs:simpleType>
          </xs:attribute>
          <xs:attribute name='push' type='xs:boolean' use='optional'
            default='true' />
          <xs:element name='namespace'
            maxOccurs='unbounded'
            use='optional'>
            <xs:complexType>
              <xs:attribute name='ns' use='required' type='xs:string'/
                >
              <xs:attribute name='type' use='required'>
                <xs:simpleType base='xs:NMTOKEN'>
                  <xs:enumeration value='none'/>
                  <xs:enumeration value='get'/>
                  <xs:enumeration value='set'/>
                  <xs:enumeration value='both'/>
                </xs:simpleType>
              </xs:attribute>
```

```
        </xs:complexType>
      </xs:element>
    </xs:complexType>
  </xs:element>
</xs:complexType>
</xs:element>

<xs:element name='privileged_iq' />

</xs:schema>
```

13 Acknowledgements