XEP-0115: Entity Capabilities

Joe Hildebrand
mailto:jhildebr@cisco.com
xmpp:hildjj@jabber.org

Peter Saint-Andre
mailto:xsf@stpeter.im
xmpp:peter@jabber.org
http://stpeter.im/

Remko Tronçon
http://el-tramo.be/

Jacek Konieczny
mailto:jajcus@jajcus.net
xmpp:jajcus@jabber.bnet.pl

2016-10-06
Version 1.5.1

Status	Type	Short Name
Draft	Standards Track	caps

This document defines an XMPP protocol extension for broadcasting and dynamically discovering client, device, or generic entity capabilities. In order to minimize network impact, the transport mechanism is standard XMPP presence broadcast (thus forestalling the need for polling related to service discovery data), the capabilities information can be cached either within a session or across sessions, and the format has been kept as small as possible.

Legal

Copyright

Permissions

Warranty

Liability

Conformance

Contents

1 Introduction

1.1 Motivation

It is often desirable for an XMPP application (commonly but not necessarily a client) to take different actions depending on the capabilities of another application from which it receives presence information. Examples include:

- Showing a different set of icons depending on the capabilities of other entities.
- Not sending XHTML-IM (XEP-0071) [1] or other rich content to plaintext clients such as cell phones.
- Allowing the initiation of a Voice over IP (VoIP) session only to clients that support Jingle (XEP-0166) [2] and Jingle RTP Sessions (XEP-0167) [3].
- Not showing a "Send a File" button if another user's client does not support SI File Transfer (XEP-0096) [4].
- Filtering Publish-Subscribe (XEP-0060) [5] notifications based on advertised subscriber interests.

In the past, after logging in some Jabber clients sent one Service Discovery (XEP-0030) [6] and one Software Version (XEP-0092) [7] request to each entity from which they received presence. That "disco/version flood" resulted in an excessive use of bandwidth and was impractical on a larger scale, particularly for users with large rosters. Therefore this document defines a more robust and scalable solution: namely, a presence-based mechanism [8] for exchanging information about entity capabilities. Clients should not engage in the older "disco/version flood" behavior and instead should use Entity Capabilities as specified herein.

1.2 How It Works

This section provides a friendly introduction to entity capabilities ("caps").
Imagine that you are a Shakespearean character named Juliet and one of your contacts, a handsome fellow named Romeo, becomes available. His client wants to publish its capabilities, and does this by adding to its presence packets a <c/> element with special attributes. As a result, your client receives the following presence packet:

[1] XEP-0071: XHTML-IM <https://xmpp.org/extensions/xep-0071.html>.
[2] XEP-0166: Jingle <https://xmpp.org/extensions/xep-0166.html>.
[3] XEP-0167: Jingle RTP Sessions <https://xmpp.org/extensions/xep-0167.html>.
[4] XEP-0096: SI File Transfer <https://xmpp.org/extensions/xep-0096.html>.
[5] XEP-0060: Publish-Subscribe <https://xmpp.org/extensions/xep-0060.html>.
[6] XEP-0030: Service Discovery <https://xmpp.org/extensions/xep-0030.html>.
[7] XEP-0092: Software Version <https://xmpp.org/extensions/xep-0092.html>.
[8] Entity capabilities is not limited to clients, and can be used by any entity that exchanges presence with another entity, e.g., a gateway. However, this specification mainly uses the example of clients.

```
<presence from='romeo@montague.lit/orchard'>
  <c xmlns='http://jabber.org/protocol/caps'
     hash='sha-1'
     node='http://code.google.com/p/exodus'
     ver='QgayPKawpkPSDYmwT/WM94uAlu0='/>
</presence>
```

The 'node' attribute represents the client software Romeo is using. The 'ver' attribute is a specially-constructed string (called a "verification string") that represents the entity's service discovery identity (category and type as registered at <https://xmpp.org/registrar/disco-categories.html>, as well as, optionally, xml:lang and name) and supported features (as registered at <https://xmpp.org/registrar/disco-features.html> as well as, optionally, extended service discovery information data registered at <https://xmpp.org/registrar/formtypes.html>).
At this point, your client has no idea what the capabilities are of someone with a verification string 'QgayPKawpkPSDYmwT/WM94uAlu0='. Your client therefore sends a service discovery query to Romeo, asking what his client can do.

```
<iq from='juliet@capulet.lit/chamber'
    id='disco1'
    to='romeo@montague.lit/orchard'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#info'
         node='http://code.google.com/p/exodus#QgayPKawpkPSDYmwT/
             WM94uAlu0='/>
</iq>
```

The response is:

```
<iq from='romeo@montague.lit/orchard'
    id='disco1'
    to='juliet@capulet.lit/chamber'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'
         node='http://code.google.com/p/exodus#QgayPKawpkPSDYmwT/
             WM94uAlu0='>
    <identity category='client' name='Exodus␣0.9.1' type='pc'/>
    <feature var='http://jabber.org/protocol/caps'/>
    <feature var='http://jabber.org/protocol/disco#info'/>
    <feature var='http://jabber.org/protocol/disco#items'/>
    <feature var='http://jabber.org/protocol/muc'/>
  </query>
</iq>
```

At this point, your client knows that a contact who advertises a verification string of 'QgayPKawpkPSDYmwT/WM94uAlu0=' supports Multi-User Chat (XEP-0045) [9] and the other features returned by Romeo because the contact in fact uses the same version of the same client software as Romeo, with the same enabled features, plugins, presented client name(s), and the like (i.e., the same input to the verification string generation method). [10] Your client remembers this information, so that it does not need to explicitly query the capabilities of a contact with the same verification string. For example, your Nurse may use the same client that Romeo does:

```
<presence from='nurse@capulet.lit/chamber'>
  <c xmlns='http://jabber.org/protocol/caps'
     hash='sha-1'
     node='http://code.google.com/p/exodus'
     ver='QgayPKawpkPSDYmwT/WM94uAlu0='/>
</presence>
```

Therefore you know that she also supports the same features that Romeo does.
On the other hand, for a person with the following presence ...

```
<presence from='benvolio@capulet.lit/230193'>
  <c xmlns='http://jabber.org/protocol/caps'
     hash='sha-1'
     node='http://psi-im.org'
     ver='q07IKJEyjvHSyhy//CH0CxmKi8w='/>
</presence>
```

... or the following presence ...

```
<presence from='bard@shakespeare.lit/globe'>
  <c xmlns='http://jabber.org/protocol/caps'
     hash='sha-1'
     node='http://www.chatopus.com'
     ver='zHyEOgxTrkpSdGcQKH8EFPLsriY='/>
</presence>
```

... you have no information about what this contact's client is capable of unless you have cached previous entity capabilities information; therefore you need to query for capabilities explicitly again via service discovery.

[9] XEP-0045: Multi-User Chat <https://xmpp.org/extensions/xep-0045.html>.
[10] The string can be relied upon because of how it is generated and checked, as explained later in this document.

2 Assumptions

This document makes several assumptions:

- The identity of the client I am using is of interest to the people in my roster.
- Clients for the people on my roster might want to make user interface decisions based on my capabilities.
- Members of a community tend to cluster around a small set of clients with a small set of capabilities. More specifically, multiple people in my roster use the same client, and they upgrade versions relatively slowly (commonly a few times a year, perhaps once a week at most, certainly not once a minute).
- Some clients are running on networks without server-to-server connectivity enabled and without access to the Internet via HTTP.
- Conversations are possible between users who are not on each other's rosters.
- Client capabilities may change over the course of a presence session, as features are enabled or disabled.

3 Requirements

The protocol defined herein addresses the following requirements:

1. Clients must be able to participate even if they support only XMPP Core [11], XMPP IM [12], and XEP-0030.
2. Clients must be able to participate even if they are on networks without connectivity to other XMPP servers, services offering specialized XMPP extensions, or HTTP servers.[13]
3. Clients must be able to retrieve information without querying every entity with which they communicate.
4. Since presence is normally broadcast to many contacts, the byte size of the proposed extension must be as small as possible.
5. It must be possible to write a XEP-0045 server implementation that passes the given information along.
6. It must be possible to publish a change in capabilities within a single presence session.

[11] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.
[12] RFC 6121: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence <http://tools.ietf.org/html/rfc6121>.
[13] These first two requirements effectively eliminated XEP-0060 as a possible implementation of entity capabilities.

7. Server infrastructure above and beyond that defined in XMPP Core and XMPP IM must not be required for this approach to work, although additional server infrastructure may be used for optimization purposes.

8. The defined mechanism must not be limited to clients but must be usable by servers, components, and other network entities.

4 Protocol

Entity capabilities are encapsulated in a <c/> element qualified by the 'http://jabber.org/protocol/caps' namespace. The attributes of the <c/> element are as follows.

Name	Definition	Inclusion
ext	A set of nametokens specifying additional feature bundles; this attribute is deprecated (see the Legacy Format section of this document).	DEPRECATED
hash	The hashing algorithm used to generate the verification string; see Mandatory-to-Implement Technologies regarding supported hashing algorithms.	REQUIRED
node	A URI that uniquely identifies a software application, typically a URL at the website of the project or company that produces the software. *	REQUIRED
ver	A string that is used to verify the identity and supported features of the entity. **	REQUIRED

* Note: It is RECOMMENDED for the value of the 'node' attribute to be an HTTP URL at which a user could find further information about the software product, such as "http://psi-im.org" for the Psi client; this enables a processing application to also determine a unique string for the generating application, which it could maintain in a list of known software implementations (e.g., associating the name received via the disco#info reply with the URL found in the caps data).
* Note: Before version 1.4 of this specification, the 'ver' attribute was used to specify the released version of the software; while the values of the 'ver' attribute that result from use of the algorithm specified herein are backwards-compatible, applications SHOULD appropriately handle the Legacy Format.

5 Verification String

5.1 Generation Method

In order to help prevent poisoning of entity capabilities information, the value of the verification string MUST be generated according to the following method.
Note: All sorting operations MUST be performed using "i;octet" collation as specified in Section 9.3 of RFC 4790 [14].

1. Initialize an empty string S.
2. Sort the service discovery identities [15] by category and then by type and then by xml:lang (if it exists), formatted as CATEGORY '/' [TYPE] '/' [LANG] '/' [NAME]. [16] Note that each slash is included even if the LANG or NAME is not included (in accordance with XEP-0030, the category and type MUST be included).
3. For each identity, append the 'category/type/lang/name' to S, followed by the '<' character.
4. Sort the supported service discovery features. [17]
5. For each feature, append the feature to S, followed by the '<' character.
6. If the service discovery information response includes XEP-0128 data forms, sort the forms by the FORM_TYPE (i.e., by the XML character data of the <value/> element).
7. For each extended service discovery information form:
   a) Append the XML character data of the FORM_TYPE field's <value/> element, followed by the '<' character.
   b) Sort the fields by the value of the "var" attribute.
   c) For each field other than FORM_TYPE:
      i. Append the value of the "var" attribute, followed by the '<' character.
      ii. Sort values by the XML character data of the <value/> element.
      iii. For each <value/> element, append the XML character data, followed by the '<' character.
8. Ensure that S is encoded according to the UTF-8 encoding (RFC 3629 [18]).

[14] RFC 4790: Internet Application Protocol Collation Registry <http://tools.ietf.org/html/rfc4790>.
[15] A registry of service discovery identities is located at <https://xmpp.org/registrar/disco-categories.html>.
[16] The combination of category, type, and xml:lang forms a unique combination, so it is not necessary to also sort by name (the name merely provides some human-readable text associated with a category/type/lang).
[17] A registry of service discovery features is located at <https://xmpp.org/registrar/disco-features.html>.
[18] RFC 3629: UTF-8, a transformation format of ISO 10646 <http://tools.ietf.org/html/rfc3629>.

9. Compute the verification string by hashing S using the algorithm specified in the 'hash' attribute (e.g., SHA-1 as defined in RFC 3174 [19]). The hashed data MUST be generated with binary output and encoded using Base64 as specified in Section 4 of RFC 4648 [20] (note: the Base64 output MUST NOT include whitespace and MUST set padding bits to zero). [21]

Note: If the four characters '&', 'l', 't', ';' appear consecutively in any of the factors of the verification string S (e.g., a service discovery identity of 'Some-Client<http://jabber.org/protocol/muc') then that string of characters MUST be treated as literally '<' and MUST NOT be converted to the character '<', because completing such a conversion would open the protocol to trivial attacks.

5.2 Simple Generation Example

Consider an entity whose category is "client", whose service discovery type is "pc", whose service discovery name is "Exodus 0.9.1", and whose supported features are "http://jabber.org/protocol/disco#info", "http://jabber.org/protocol/disco#items", and "http://jabber.org/protocol/muc". Using the SHA-1 algorithm, the verification string would be generated as follows (note: line breaks in the verification string are included only for the purposes of readability):

1. S = "
2. Only one identity: "client/pc"
3. S = 'client/pc//Exodus 0.9.1<'
4. Sort the features: "http://jabber.org/protocol/caps", "http://jabber.org/protocol/disco#info", "http://jabber.org/protocol/disco#items", "http://jabber.org/protocol/muc".
5. S = 'client/pc//Exodus 0.9.1<http://jabber.org/protocol/caps<http://jabber.org/protocol/disco#info<http://jabber.org/protocol/disco#items<http://jabber.org/protocol/muc<'
6. ver = QgayPKawpkPSDYmwT/WM94uAlu0=

[19] RFC 3174: US Secure Hash Algorithm 1 (SHA1) <http://tools.ietf.org/html/rfc3174>.
[20] RFC 4648: The Base16, Base32, and Base64 Data Encodings <http://tools.ietf.org/html/rfc4648>.
[21] The OpenSSL command for producing such output with SHA-1 is "echo -n 'S' | openssl dgst -binary -sha1 | openssl enc -nopad -base64".

5.3 Complex Generation Example

Consider a more complex example, where the entity includes several identities (with the service discovery name in different languages) as well as extended information formatted according to XEP-0128.

```
<iq from='benvolio@capulet.lit/230193'
    id='disco1'
    to='juliet@capulet.lit/chamber'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'
         node='http://psi-im.org#q07IKJEyjvHSyhy//CH0CxmKi8w='>
    <identity xml:lang='en' category='client' name='Psi 0.11' type='pc
        '/>
    <identity xml:lang='el' category='client' name=' 0.11' type='pc'/
        >
    <feature var='http://jabber.org/protocol/caps'/>
    <feature var='http://jabber.org/protocol/disco#info'/>
    <feature var='http://jabber.org/protocol/disco#items'/>
    <feature var='http://jabber.org/protocol/muc'/>
    <x xmlns='jabber:x:data' type='result'>
      <field var='FORM_TYPE' type='hidden'>
        <value>urn:xmpp:dataforms:softwareinfo</value>
      </field>
      <field var='ip_version'>
        <value>ipv4</value>
        <value>ipv6</value>
      </field>
      <field var='os'>
        <value>Mac</value>
      </field>
      <field var='os_version'>
        <value>10.5.1</value>
      </field>
      <field var='software'>
        <value>Psi</value>
      </field>
      <field var='software_version'>
        <value>0.11</value>
      </field>
    </x>
  </query>
</iq>
```

Using the SHA-1 algorithm, the verification string would be generated as follows (note: line breaks in the verification string are included only for the purposes of readability):

1. S = ”

2. Two identities: ”client/pc/Psi” and ”client/pc/”

3. S = ’client/pc/el/ 0.11<client/pc/en/Psi 0.11<’

4. Sort the features: ”http://jabber.org/protocol/caps”, http://jabber.org/protocol/disco#info”, ”http://jabber.org/protocol/disco#items”, ”http://jabber.org/protocol/muc”.

5. S = ’client/pc/el/ 0.11<client/pc/en/Psi 0.11<http://jabber.org/protocol/caps<http://jabber.org/protocol http://jabber.org/protocol/disco#items<http://jabber.org/protocol/muc<’.

6. Sort the extended service discovery forms by FORM_TYPE (there is only one: ”urn:xmpp:dataforms:softwareinfo”).

7. S = ’client/pc/el/ 0.11<client/pc/en/Psi 0.11<http://jabber.org/protocol/caps<http://jabber.org/protocol http://jabber.org/protocol/disco#items<http://jabber.org/protocol/muc<urn:xmpp:dataforms:softwareinf

8. Sort the fields by var and append the value(s): ”ip_version<ipv4<ipv6”, ”os<Mac”, ”os_version<10.5.1”, ”software<Psi”, ”software_version<0.11”.

9. S = ’client/pc/el/ 0.11<client/pc/en/Psi 0.11<http://jabber.org/protocol/caps<http://jabber.org/protocol http://jabber.org/protocol/disco#items<http://jabber.org/protocol/muc<urn:xmpp:dataforms:softwareinf ip_version<ipv4<ipv6<os<Mac<os_version<10.5.1<software<Psi<software_version<0.11<’

10. ver = q07IKJEyjvHSyhy//CH0CxmKi8w=

5.4 Processing Method

When an entity receives a value of the ’ver’ attribute that appears to be a verification string generated in accordance with the generation method defined in this specification, it MUST process the ’ver’ according to the following method.

1. Verify that the <c/> element includes a ’hash’ attribute. If it does not, ignore the ’ver’ or treat it as generated in accordance with the Legacy Format (if supported).

2. If the value of the 'hash' attribute does not match one of the processing application's supported hash functions, do the following:
   a) Send a service discovery information request to the generating entity.
   b) Receive a service discovery information response from the generating entity.
   c) Do not validate or globally cache the verification string as described below; instead, the processing application SHOULD associate the discovered identity+features *only* with the JabberID of the generating entity.
3. If the value of the 'hash' attribute matches one of the processing application's supported hash functions, validate the verification string by doing the following:
   a) Send a service discovery information request to the generating entity.
   b) Receive a service discovery information response from the generating entity.
   c) If the response includes more than one service discovery identity with the same category/type/lang/name, consider the entire response to be ill-formed.
   d) If the response includes more than one service discovery feature with the same XML character data, consider the entire response to be ill-formed.
   e) If the response includes more than one extended service discovery information form with the same FORM_TYPE or the FORM_TYPE field contains more than one <value/> element with different XML character data, consider the entire response to be ill-formed.
   f) If the response includes an extended service discovery information form where the FORM_TYPE field is not of type "hidden" or the form does not include a FORM_TYPE field, ignore the form but continue processing.
   g) If the response is considered well-formed, reconstruct the hash by using the service discovery information response to generate a local hash in accordance with the Generation Method).
   h) If the values of the received and reconstructed hashes match, the processing application MUST consider the result to be valid and SHOULD globally cache the result for all JabberIDs with which it communicates.
   i) If the values of the received and reconstructed hashes do not match, the processing application MUST consider the result to be invalid and MUST NOT globally cache the verification string; however, it SHOULD check the service discovery identity and supported features of another generating entity who advertises that value.

Note: If the four characters '&', 'l', 't', ';' appear consecutively in any of the factors of the verification string S (e.g., a service discovery identity of 'SomeClient<http://jabber.org/protocol/muc') then that string of characters MUST be treated as literally '<' and MUST NOT be converted to the character '<', because completing such a conversion would open the protocol to trivial attacks.

6 Use Cases

6.1 Advertising Capabilities

Each time a generating entity sends presence, it annotates that presence with an entity identifier ('node' attribute) and identity and feature identifier ('ver' attribute). So that servers can remember the last presence for use in responding to probes, a client SHOULD include entity capabilities with every presence notification it sends.

Listing 1: Presence with caps

```
<presence>
  <c xmlns='http://jabber.org/protocol/caps'
     hash='sha-1'
     node='http://code.google.com/p/exodus'
     ver='QgayPKawpkPSDYmwT/WM94uAlu0='/>
</presence>
```

If the supported features change during a generating entity's presence session (e.g., a user installs an updated version of a client plugin), the application MUST recompute the verification string and SHOULD send a new presence broadcast.

Listing 2: Presence with recomputed ver attribute

```
<presence>
  <c xmlns='http://jabber.org/protocol/caps'
     hash='sha-1'
     node='http://code.google.com/p/exodus'
     ver='66/0NaeaBKkwk85efJTGmU47vXI='/>
</presence>
```

6.2 Discovering Capabilities

An application (the "requesting entity") can learn what features another entity supports by sending a disco#info request (see XEP-0030) to the entity that generated the caps information (the "generating entity").

Listing 3: Disco#info request

```
<iq from='juliet@capulet.lit/balcony'
    id='disco1'
    to='romeo@montague.lit/orchard'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#info'
         node='http://code.google.com/p/exodus#QgayPKawpkPSDYmwT/
             WM94uAlu0='/>
</iq>
```

The disco#info request is sent by the requesting entity to the generating entity. The value of the 'to' attribute MUST be the exact JID of the generating entity, which in the case of a client will be the full JID <localpart@domain.tld/resource>.
Note: The generating entity SHOULD NOT include the "caps node" in the list of entities it returns in its disco#items responses; i.e., the caps node is a kind of virtual or phantom node, not a true items node that is associated with the generating entity for service discovery purposes.
The disco 'node' attribute MUST be included for backwards-compatibility. The value of the 'node' attribute SHOULD be generated by concatenating the value of the caps 'node' attribute (e.g., "http://code.google.com/p/exodus") as provided by the generating entity, the "#" character, and the value of the caps 'ver' attribute (e.g., "QgayPKawpkPSDYmwT/WM94uAlu0=") as provided by the generating entity.
The generating entity then returns all of the capabilities it supports.

Listing 4: Disco#info response

```
<iq from='romeo@montague.lit/orchard'
    id='disco1'
    to='juliet@capulet.lit/balcony'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'
         node='http://code.google.com/p/exodus#QgayPKawpkPSDYmwT/
             WM94uAlu0='>
    <identity category='client' type='pc'/>
    <feature var='http://jabber.org/protocol/disco#info'/>
    <feature var='http://jabber.org/protocol/disco#items'/>
    <feature var='http://jabber.org/protocol/muc'/>
  </query>
</iq>
```

Note: If the generating entity incorporated multiple identities with different xml:lang values in its verification string, it MUST return all of the identities even if the request specified a particular xml:lang.

6.3 Stream Feature

A server MAY include its entity capabilities in a stream feature element so that connecting clients and peer servers do not need to send service discovery requests each time they connect.

Listing 5: Stream feature element including capabilities

```
<stream:features>
  <c xmlns='http://jabber.org/protocol/caps'
     hash='sha-1'
     node='http://jabberd.org'
     ver='ItBTI0XLDFvVxZ72NQElAzKS9sU='/>
```

```
</stream:features>
```

When a connected client or peer server sends a service discovery information request to determine the entity capabilities of a server that advertises capabilities via the stream feature, the requesting entity MUST send the disco#info request to the server's JID as provided in the 'from' attribute of the response stream header (the 'from' attribute was recommended by RFC 3920 [22] and is required by RFC 6120 [23]). To enable this functionality, a server that advertises support for entity capabilities MUST provide a 'from' address in its response stream headers, in accordance with RFC 6120.

7 Determining Support

If an entity supports the entity capabilities protocol, it MUST advertise that fact by returning a feature of **'http://jabber.org/protocol/caps'** in response to a service discovery information request.

Listing 6: Service discovery information request

```
<iq from='romeo@montague.lit/orchard'
    id='disco2'
    to='juliet@capulet.lit/balcony'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

Listing 7: Service discovery information response

```
<iq from='juliet@capulet.lit/balcony'
    id='disco2'
    to='romeo@montague.lit/orchard'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    ...
    <feature var='http://jabber.org/protocol/caps'/>
    ...
  </query>
</iq>
```

If a server supports the Caps Optimization functionality, it MUST also return a feature of **'http://jabber.org/protocol/caps#optimize'** in response to service discovery information requests.

[22] RFC 3920: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc3920>.
[23] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.

Listing 8: Service discovery information request

```
<iq from='juliet@capulet.lit/balcony'
    id='disco3'
    to='capulet.lit'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

Listing 9: Service discovery information response

```
<iq from='capulet.lit'
    id='disco3'
    to='juliet@capulet.lit/balcony'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    ...
    <feature var='http://jabber.org/protocol/caps#optimize'/>
    ...
  </query>
</iq>
```

8 Implementation Notes

8.1 Hashing Algorithm Support

An application SHOULD maintain a list of hashing algorithms it supports, which MUST include the algorithm or algorithms listed in the Mandatory-to-Implement Technologies section of this document.

8.2 Caching

It is RECOMMENDED for an application that processes entity capabilities information to cache associations between the verification string and discovered identity+features within the scope of one presence session. This obviates the need for extensive service discovery requests within a session.
It is RECOMMENDED for an application to cache associations across presence sessions, since this obviates the need for extensive service discovery requests at the beginning of a session (this is especially helpful in bandwidth-constrained environments).

8.3 Directed Presence

If two entities exchange messages but they do not normally exchange presence (i.e., via presence subscription), the entities MAY choose to send directed presence to each other,

where the presence information SHOULD be annotated with the same capabilities information as each entity sends in presence broadcasts. Until and unless capabilities information has been received from another entity, an application MUST assume that the other entity does not support capabilities.

8.4 Caps Optimization

A server that is managing an connected client's presence session MAY optimize presence notification traffic sent through the server by stripping off redundant capabilities annotations (i.e., the <c/> element). Because of this, receivers of presence notifications MUST NOT expect an annotation on every presence notification they receive. If the server performs caps optimization, it MUST ensure that the first presence notification each subscriber receives contains the annotation. The server MUST also ensure that any changes in the caps information (e.g., an updated 'ver' attribute) are sent to all subscribers.
If a connected client determines that its server supports caps optimization, it MAY choose to send the capabilities annotation only on the first presence packet, as well as whenever its capabilities change.

9 Security Considerations

9.1 Mandatory-to-Implement Technologies

The SHA-1 hashing algorithm is mandatory to implement. All implementations MUST support SHA-1.
An implementation MAY support other algorithms. Any such algorithm SHOULD be registered in the IANA Hash Function Textual Names Registry [24].
In the future, the XMPP Council [25] may, at its discretion, modify the mandatory-to-implement hashing algorithm if it determines that SHA-1 has become practically vulnerable to Preimage Attacks.

9.2 Preimage Attacks

As described in RFC 4270 [26], protocols that use the output of hash functions such as MD5 or SHA-1 can be vulnerable to collision attacks or preimage attacks or both. Because of how the hash output is used in entity capabilities, the protocol will not be subject to collision attacks even if the hash function used is found to be vulnerable to collision attacks. However, it is

[24] IANA registry of Hash Function Textual Names <http://www.iana.org/assignments/hash-function-text-names>.
[25] The XMPP Council is a technical steering committee, authorized by the XSF Board of Directors and elected by XSF members, that approves of new XMPP Extensions Protocols and oversees the XSF's standards process. For further information, see <https://xmpp.org/about/xmpp-standards-foundation#council>.
[26] RFC 4270: Attacks on Cryptographic Hashes in Internet Protocols <http://tools.ietf.org/html/rfc4270>.

possible that the protocol might become subject to preimage attacks if the hash function used is found to be vulnerable to preimage attacks.
In theory, such a preimage attack would take one of the following forms:

- Given knowledge of a particular value V of the 'ver' attribute, an attacker can find an input message X such that hash(X) yields V (this is known as a "first preimage attack").
- Given knowledge of a particular value S used as the input message to the hash function, an attacker can find a value S' that yields V (this is known as a "second preimage attack").

In practice, a preimage attack would need to meet all of the following criteria in order to be effective against the entity capabilities protocol:

1. The hashing algorithm used would need to be found not only theoretically but practically vulnerable to first or second preimage attacks (e.g., this is not yet true of the MD5 or SHA-1 algorithms, but may become true in the future).
2. An attacker would need to find an input message X or S' that matches the hash V for a particular value of V or S, which may not be practical given that (a) the values of S used as input to the hash function in entity capabilities are relatively short and (b) cryptanalysis to date indicates that existing hash functions may not be vulnerable to preimage attacks except in the case of relatively long input messages (on the order of 2^{55} blocks).
3. The input message X or S' would need to conform to the structure of S as specified under Verification String, including the order of service discovery identity or identities followed by service discovery features, delimited by the '<' character and sorted using "i;octet" collation.
4. The input messsage X or S' would need to make it seem as if a desirable feature (e.g., end-to-end encryption) is not supported by other entities that advertise the same hash V even though the feature is indeed supported (i.e., the attacker would need to return a set of service discovery identities and features that match X or S', and have that set be plausible for an entity that communicates via XMPP), or make it seem as if an undesirable feature is supported even though the feature is not supported.
5. The attacker would need to propagate the hash V before some other entity with the true input message S could broadcast presence with the relevant entity capabilities data and provide the true service discovery response (thus the attacker might need to subvert the development process of a particular software project or subvert the namespace issuance process of the XMPP Registrar [27], or both).

[27] 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/>.

It currently seems extremely unlikely that an attacker could meet all of the foregoing conditions in the foreseeable future. However, the XMPP Council shall continue to monitor the state of cryptanalysis regarding the mandatory-to-implement hash function as well as the possibility that any vulnerabilities in that function might lead to practical threats against the entity capabilities protocol. If and when it becomes practical (or even possible) to launch effective preimage attacks against the entity capabilities protocol, the XMPP Council shall consider updating this specification to change the mandatory-to-implement hashing algorithm to a safer technology.
Note: If the four characters '&', 'l', 't', ';' appear consecutively in any of the factors of the verification string S (e.g., a service discovery identity of 'Some-Client<http://jabber.org/protocol/muc') then that string of characters MUST be treated as literally '<' and MUST NOT be converted to the character '<', because completing such a conversion would open the protocol to trivial attacks.

9.3 Caps Poisoning

Adherence to the method defined in the Verification String section of this document for both generation and processing of the 'ver' attribute helps to guard against poisoning of entity capabilities information by malicious or improperly implemented entities.
If the value of the 'ver' attribute is a verification string as defined herein (i.e., if the 'ver' attribute is not generated according to the Legacy Format), inclusion of the 'hash' attribute is REQUIRED. Knowing explicitly that the value of the 'ver' attribute is a verification string enables the recipient to avoid spurious notification of invalid or poisoned hashes.

9.4 Information Exposure

Use of entity capabilities might make it easier for an attacker to launch certain application-specific attacks, since the attacker could more easily determine the type of client being used as well as its capabilities. However, since most clients respond to Service Discovery and Software Version requests without performing access control checks, there is no new vulnerability. Entities that wish to restrict access to capabilities information SHOULD use Privacy Lists (XEP-0016) [28] to define appropriate communications blocking (e.g., an entity MAY choose to allow IQ requests only from "trusted" entities, such as those with whom it has a presence subscription of "both"); note, however, that such restrictions may be incompatible with the recommendation regarding Directed Presence.

[28] XEP-0016: Privacy Lists <https://xmpp.org/extensions/xep-0016.html>.

10 IANA Considerations

This document requires no interaction with the Internet Assigned Numbers Authority (IANA) [29].

11 XMPP Registrar Considerations

11.1 Protocol Namespaces

The XMPP Registrar [30] includes "http://jabber.org/protocol/caps" in its registry of protocol namespaces (see <https://xmpp.org/registrar/namespaces.html>).

11.2 Service Discovery Features

The XMPP Registrar includes "http://jabber.org/protocol/caps" and "http://jabber.org/protocol/caps#optimize" in its registry of service discovery features (see <https://xmpp.org/registrar/disco-features.html>).

11.3 Stream Features

The XMPP Registrar includes "http://jabber.org/protocol/caps" in its registry of stream features (see <https://xmpp.org/registrar/stream-features.html>).

12 XML Schema

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

<xs:schema
    xmlns:xs='http://www.w3.org/2001/XMLSchema'
    targetNamespace='http://jabber.org/protocol/caps'
    xmlns='http://jabber.org/protocol/caps'
    elementFormDefault='qualified'>

  <xs:annotation>
    <xs:documentation>
```

[29] 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/>.

[30] 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/>.

```
      The protocol documented by this schema is defined in
      XEP-0115: http://www.xmpp.org/extensions/xep-0115.html
    </xs:documentation>
  </xs:annotation>

  <xs:element name='c'>
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base='empty'>
          <xs:attribute name='ext' type='xs:NMTOKENS' use='optional'/>
          <xs:attribute name='hash' type='xs:NMTOKEN' use='required'/>
          <xs:attribute name='node' type='xs:string' use='required'/>
          <xs:attribute name='ver' type='xs:string' use='required'/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>

  <xs:simpleType name='empty'>
    <xs:restriction base='xs:string'>
      <xs:enumeration value=''/>
    </xs:restriction>
  </xs:simpleType>

</xs:schema>
```

13 Legacy Format

Before Version 1.4 of this specification, the 'ver' attribute was generated differently, the 'ext' attribute was used more extensively, and the 'hash' attribute was absent. For historical purposes, Version 1.3 of this specification is archived at <http://www.xmpp.org/extensions/attic/xep-0115-1.3.html>. For backwards-compatibility with the legacy format, the 'node' attribute is REQUIRED and the 'ext' attribute MAY be included.

An application can determine if the legacy format is in use by checking for the presence of the 'hash' attribute, which is REQUIRED in the current format.

If a caps-processing application supports the legacy format, it SHOULD check the 'node', 'ver', and 'ext' combinations as specified in the archived version 1.3 of this specification, and MAY cache the results.

If a caps-processing application does not support the legacy format, it SHOULD ignore the 'ver' value entirely (since the value cannot be verified) and SHOULD NOT cache it, since the application cannot validate the identity and features by checking the hash.

14 Acknowledgements

Thanks to Rachel Blackman, Dave Cridland, Richard Dobson, Olivier Goffart, Sergei Golovan, Justin Karneges, Ralph Meijer, Ian Paterson, Kevin Smith, Tomasz Sterna, Michal Vaner, and Matt Yacobucci for comments and suggestions.