XEP-0025: Jabber HTTP Polling

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

Craig Kaes
mailto:ckaes@jabber.com
xmpp:ckaes@corp.jabber.com

David Waite
mailto:mass@akuma.org
xmpp:mass@akuma.org

2009-06-03
Version 1.2

Status	Type	Short Name
Obsolete	Historical	httppoll

This document defines an XMPP protocol extension that enables access to a Jabber server from behind firewalls which do not allow outgoing sockets on port 5222, via HTTP requests.

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

Note Well: This protocol specified in this document has been superseded by the protocol specified in BOSH (XEP-0124) [1].

This specification documents a method to allow Jabber clients to access Jabber servers from behind existing firewalls. Although several similar methods have been proposed, this approach should work through all known firewall configurations which allow outbound HTTP access.

2 Background

In general, a firewall is a box that protects a network from outsiders, by controlling the IP connections that are allowed to pass through the box. Often, a firewall will also allow access outside only by proxy, either explicit proxy support or implicit through Network Address Translation (NAT).

In the interest of security, many firewall administrators do not allow outbound connections to unknown and unused ports. Until Jabber becomes more widely deployed, port 5222/tcp (for Jabber client connections) will often be blocked.

The best solution for sites that are concerned about security is to run their own Jabber server, either inside the firewall, or in a DMZ [2] network. However, there are network configuration where an external Jabber server must still be used and port 5222/tcp outbound cannot be allowed. In these situations, different methods for connecting to a Jabber server are required.

Several methods exist today for doing this traversal. Most rely on the fact that a most firewalls are configured to allow access through port 80/tcp. Although some less-complicated firewalls will allow any protocol to traverse this port, many will proxy, filter, and verify requests on this port as HTTP. Because of this, a normal Jabber connection on port 80/tcp will not suffice. In addition, many firewalls/proxy servers will also not allow or not honor HTTP Keep-alives (as defined in section 19.7.1.1 of RFC 2068 [3]) and will consider long-lived socket connections as security issues. Because of this the traditional Jabber connection model, where one socket is one stream is one session, will not work reliably.

In light of all of the ways that default firewall rules can interfere with Jabber connectivity, a lowest-common denominator approach was selected. HTTP is used to send XML as POST requests and receieve pending XML within the responses. Additional information is prepended in the request body to ensure an equivalent level of security to TCP/IP sockets.

[1] XEP-0124: Bidirectional-streams Over Synchronous HTTP <https://xmpp.org/extensions/xep-0124.html>.
[2] DMZ definition at searchwebmanagement.com
[3] RFC 2068: Hypertext Transport Protocol -- HTTP/1.1 <http://tools.ietf.org/html/rfc2068>.

3 Normal data transfer

The client makes HTTP requests periodically to the server. Whenever the client has something to send, that XML is included in the body of the request. When the server has something to send to the client, it must be contained in the body of the response.
In some browser/platform combinations, sending cookies from the client is not possible due to design choices and limitations in the browser. Therefore, a work-around was needed to support clients based on these application platforms.
All requests to the server are HTTP POST requests, with Content-Type: application/x-www-form-urlencoded. Responses from the server have Content-Type: text/xml. Both the request and response bodies are UTF-8 encoded text, even if an HTTP header to the contrary exists. All responses contain a Set-Cookie header with an identifier, which is sent along with future requests as described below. This identifier cookie must have a name of 'ID'. The first request to a server always uses 0 as the identifier. The server must always return a 200 response code, sending any session errors as specially-formatted identifiers.
The client sends requests with bodies in the following format:

```
identifier ; key [ ; new_key] , [xml_body]
```

If the identifier is zero, key indicates an initial key. In this case, new_key should not be specified, and must be ignored.

Identifier	Purpose
identifier	To uniquely identify the session server-side. This field is only used to identify the session, and provides no security.
key	To verify this request is from the originator of the session. The client generates a new key in the manner described below for each request, which the server then verifies before processing the request.
new_key	The key algorithm can exhaust valid keys in a sequence, which requires a new key sequence to be used in order to continue the session. The new key is sent along with the last used key in the old sequence.
xml_body	The body of text to send. Since a POST must be sent in order for the server to respond with recent messages, a client may send a request without an xml_-body in order to just retrieve new incoming packets. This is not required to be a full XML document or XML fragment, it does not need to start or end on element boundaries.

The identifier is everything before the first semicolon, and must consist of the characters [A-Za-z0-9:-]. The identifier returned from the first request is the identifier for the session. Any new identifier that ends in ':0' indicates an error, with the entire identifier indicating the specific error condition. Any new identifier that does not end in ':0' is a server programming error, the client should discontinue the session. For new sessions, the client identifier is

considered to be 0.

3.1 Error conditions

Any identifier that ends in ':0' indicates an error. Any previous identifier associated with this session is no longer valid.

3.1.1 Unknown Error

Server returns ID=0:0. The response body can contain a textual error message.

3.1.2 Server Error

Server returns ID=-1:0

3.1.3 Bad Request

Server returns ID=-2:0

3.1.4 Key Sequence Error

Server returns ID=-3:0
The key is a client security feature to allow TCP/IP socket equivalent security. It does not protect against intermediary attacks, but does prevent a person who is capable of listening to the HTTP traffic from sending messages and receiving incoming traffic from another machine. The key algorithm should be familiar with those with knowledge of Jabber zero-knowledge authentication.

```
K(n, seed) = Base64Encode(SHA1(K(n - 1, seed))), for n > 0
        K(0, seed) = seed, which is client-determined
```

Note: Base64 encoding is defined in RFC 3548 [4]. SHA1 is defined in RFC 3174 [5].
No framing is implied by a single request or reply. A single request can have no content sent, in which case the body contains only the identifier followed by a comma. A reply may have no content to send, in which case the body is empty. Zero or more XMPP packets may be sent in a single request or reply, including partial XMPP packets.
The absense of a long-lived connection requires the server to consider client traffic as a heartbeat to keep the session alive. If a server-configurable period of time passes without a

[4] RFC 3548: The Base16, Base32, and Base64 Data Encodings <http://tools.ietf.org/html/rfc3548>.
[5] RFC 3174: US Secure Hash Algorithm 1 (SHA1) <http://tools.ietf.org/html/rfc3174>.

successful POST request sent by the client, the server must end the client session. Any client requests using the identifier associated with that now dead session must return an error of '0:0'.
The maximum period of time to keep a client session active without an incoming POST request is not defined, but five minutes is the recommended minimum. The maximum period of time recommended for clients between requests is two minutes; if the client has not sent any XML out for two minutes, a request without an XML body should be sent. If a client is disconnecting from the server, a closing <stream:stream> must be sent to end the session. Failure to do this may have the client continue to be represented to other users as available.
If the server disconnects the user do to a session timeout, the server MUST bounce pending IQ requests and either bounce or store offline incoming messages.

4 Usage

The following is the sequence used for client communication:

1. The client generates some initial K(0, seed) and runs the algorithm above 'n' times to determine the initial key sent to the server, K(n, seed)
2. The client sends the request to the server to start the stream, including an identifier with a value of zero and K(n, seed)
3. The server responds with the session identifier in the headers (within the Set-Cookie field).
4. For each further request done by the client, the identifier from the server and K(n - 1, seed) are sent along.
5. The server verifies the incoming value by generating K(1, incoming_value), and verifying that value against the value sent along with the last client request. If the values do not match, the request should be ignored or logged, with an error code being returned of -3:0. The request must not be processed, and must not extend the session keepalive.
6. The client may send a new key K(m, seed') at any point, but should do this for n > 0 and must do this for n = 0. If K(0, seed) is sent without a new key, the client will not be able to continue the session.

Listing 1: Initial request (without keys)

```
POST /wc12/webclient HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: webim.jabber.com

0,<stream:stream to="jabber.com"
  xmlns="jabber:client"
  xmlns:stream="http://etherx.jabber.org/streams">
```

Listing 2: Initial response

```
Date: Fri, 15 Mar 2002 20:30:30 GMT
Server: Apache/1.3.20
Set-Cookie: ID=7776:2054; path=/webclient/; expires=-1
Content-Type: text/xml

<?xml version='1.0'?>
<stream:stream xmlns:stream='http://etherx.jabber.org/streams'
  id='3C9258BB'
  xmlns='jabber:client' from='jabber.com'>
```

Listing 3: Next request (without keys)

```
POST /wc12/webclient HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: webim.jabber.com

7776:2054,<iq type="get" id="WEBCLIENT3">
  <query xmlns="jabber:iq:auth">
    <username>hildjj</username>
  </query>
</iq>
```

Listing 4: key sequence

```
K(0, "foo") = "foo"
      K(1, "foo") = "C+7Hteo/D9vJXQ3UfzxbwnXaijM="
      K(2, "foo") = "6UU8CDmH3O4aHFmCqSORCn721+M="
      K(3, "foo") = "vFFYSOhGyaGUgLrldtMBX7x91Wc="
      K(4, "foo") = "ZaDxCilBVTHS9dJfbBo1NsC2b+8="
      K(5, "foo") = "moPFsvHytDGiJQOjp186AMXAeP0="
      K(6, "foo") = "VvxEk07IFy6hUmG/PPBlTLE2fiA="
```

Listing 5: Initial request (with keys)

```
POST /wc12/webclient HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: webim.jabber.com

0;VvxEk07IFy6hUmG/PPBlTLE2fiA=,<stream:stream to="jabber.com"
  xmlns="jabber:client"
  xmlns:stream="http://etherx.jabber.org/streams">
```

Listing 6: Next request (with keys)

```
POST /wc12/webclient HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: webim.jabber.com
```

```
7776:2054;moPFsvHytDGiJQOjp186AMXAeP0=,<iq type="get" id="WEBCLIENT3">
  <query xmlns="jabber:iq:auth">
    <username>hildjj</username>
  </query>
</iq>
```

Listing 7: Changing key

```
POST /wc12/webclient HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: webim.jabber.com

7776:2054;C+7Hteo/D9vJXQ3UfzxbwnXaijM=;Tr697Eff02+32FZp38Xaq2+3Bv4=,<
    presence/>
```

5 Known issues

- This method works over HTTPS, which is good from the standpoint of functionality, but bad in the sense that a massive amount of hardware would be needed to support reasonable polling intervals for non-trivial numbers of clients.