XEP-0440: SASL Channel-Binding Type Capability

Florian Schmaus
mailto:flo@geekplace.eu
xmpp:flo@geekplace.eu

Thilo Molitor
mailto:thilo+xmpp@eightysoft.de
xmpp:thilo.molitor@juforum.de

2025-12-08
Version 1.0.0

Status	Type	Short Name
Draft	Standards Track	sasl-cb-types

This specification allows servers to annouce their supported SASL channel-binding types to clients.

Legal

Copyright

Permissions

Warranty

Liability

Conformance

Contents

1 Introduction

SASL channel-binding is a technique to increase the security of connections (RFC 5056 [1]). Unfortunately, the SASL profile specified in RFC 6120 [2] lacks a method for the server to announce its supported channel-binding types. This hinders the adoption of channel-binding, especially since the error protocol to execute after a client requested a channel-binding type unsupported by the server is basically unspecified.
The extension defined herein fills the gap left by RFC 6120 [3], by allowing the server to announce its supported channel-binding types.

2 Announcing the SASL Channel-Binding Type Capability

This protocol consists of a stream feature named <sasl-channel-binding/> qualified by the 'urn:xmpp:sasl-cb:0' namespace. The <sasl-channel-binding/> element MUST contain one or more <channel-binding/> elements, of which each MUST have an attribute with the name 'type'. The value of the 'type' attribute SHOULD be the "Channel-binding unique prefix" of a channel-binding type which was registered with the IANA Channel-Binding Types Registry [4].
A server declares that it supports particular channel-binding types by listing the supported types via the <sasl-channel-binding/> stream feature defined herein. The <sasl-channel-binding/> element could appear next to the SASL <mechanisms/> stream-feature element, qualified by the 'urn:ietf:params:xml:ns:xmpp-sasl' namespace, as specified in RFC 6120 [5]. Another potential appearance of <sasl-channel-binding/> is next to the <authentication/> stream-feature element as specified in the Extensible SASL Profile (XEP-0388) [6].

Listing 1: Example SASL1 <mechanisms/> stream feature with SASL Channel-Binding Type Capability.

```
<stream:features>
  <sasl-channel-binding xmlns='urn:xmpp:sasl-cb:0'>
    <channel-binding type='tls-server-end-point'/>
    <channel-binding type='tls-exporter'/>
  </sasl-channel-binding>
  <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
    <mechanism>EXTERNAL</mechanism>
    <mechanism>SCRAM-SHA-1-PLUS</mechanism>
    <mechanism>PLAIN</mechanism>
  </mechanisms>
</stream:features>
```

[1] RFC 5056: On the Use of Channel Bindings to Secure Channels <http://tools.ietf.org/html/rfc5056>.
[2] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.
[3] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.
[4] IANA Channel-Binding Types Registry <https://www.iana.org/assignments/channel-binding-types/channel-binding-types.xhtml>.
[5] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.
[6] XEP-0388: Extensible SASL Profile <https://xmpp.org/extensions/xep-0388.html>.

Listing 2: Example SASL2 <authentication> stream feature with SASL Channel-Binding Type Capability.

```
<stream:features>
  <sasl-channel-binding xmlns='urn:xmpp:sasl-cb:0'>
    <channel-binding type='tls-server-end-point'/>
    <channel-binding type='tls-exporter'/>
  </sasl-channel-binding>
  <authentication xmlns='urn:xmpp:sasl:2'>
    <mechanism>SCRAM-SHA-1</mechanism>
    <mechanism>SCRAM-SHA-1-PLUS</mechanism>
    <inline>
      <sm xmlns='urn:xmpp:sm:3'/>
      <bind xmlns='urn:xmpp:bind:0'/>
    </inline>
  </authentication>
</stream:features>
```

3 Business Rules

Client developers MUST follow the following rules to ensure that implementing this specification in conjunction with SCRAM (RFC 5802 [7]) or any other equivalent SASL mechanism supporting channel-binding does not introduce a MITM attack vector. The rules are specifically taylored to SCRAM. For other SASL mechanisms supporting channel-binding, client developers MUST use the equivalent to "y" and "n" defined by those mechanisms.

1. Servers MUST implement tls-server-end-point and enable/advertise it. Clients SHOULD implement tls-server-end-point and use it if no stronger channel-binding method is mutually supported.
2. If the client doesn't support channel-binding, it MUST send "n" in the GSS-header (see RFC 5802 [8]).
3. If the client supports channel-binding, but the server announced neither SASL mechanisms supporting channel-binding nor channel-binding types (via this specification), it MUST send "y" in accordance to RFC 5802 [9].
4. If the client is using SASL2 (Extensible SASL Profile (XEP-0388) [10]) to authenticate and received announcements for SASL mechanisms supporting channel-binding from the

[7] RFC 5802: Salted Challenge Response Authentication Mechanism (SCRAM) SASL and GSS-API Mechanisms <http://tools.ietf.org/html/rfc5802>.
[8] RFC 5802: Salted Challenge Response Authentication Mechanism (SCRAM) SASL and GSS-API Mechanisms <http://tools.ietf.org/html/rfc5802>.
[9] RFC 5802: Salted Challenge Response Authentication Mechanism (SCRAM) SASL and GSS-API Mechanisms <http://tools.ietf.org/html/rfc5802>.
[10] XEP-0388: Extensible SASL Profile <https://xmpp.org/extensions/xep-0388.html>.

server, but no channel-binding types, it MUST abort the authentication (this is either an implementation error or ongoing MITM attack where channel-binding types are stripped). This condition is undefined with SASL1 (RFC 6120 [11]), but the client MAY also abort authentication in this case (or it MAY try authentication using tls-server-end-point or any other channel-binding mechanism).

5. If the client is using SASL2 to authenticate and it receives channel-binding announcements, from the server, but no SASL mechanisms supporting channel-binding are announced, it MUST abort authentication (this is an implementation error or ongoing MITM attack where SASL mechanisms are stripped). It SHOULD do the same when using SASL1.

6. If the client is using either SASL1 or SASL2 to authenticate and receives announcements for SASL mechanisms supporting channel-binding and channel-binding types, but none of these announced channel-binding types are supported by it, it MUST abort authentication, if tls-server-end-point is also not advertised by the server (this is an ongoing MITM attack, as per our first rule above).

7. The client SHOULD use a channel-binding stronger than tls-server-end-point, if advertised by the server and implemented by it (that is: set the p=<channel-binding-name> GSS-header to that channel-binding type according to RFC 5802 [12]).

When implementing SASL SCRAM Downgrade Protection (XEP-0474) [13] the above rules might be relaxed a bit, see that specification for details.
A more sophisticated attacker managing to steal the certificate and private key of the server won't be detected by the tls-server-end-point channel-binding. Clients and servers SHOULD implement stronger channel-binding types like tls-exporter, to detect and prevent such attacks

4 Security Considerations

The following considerations refer to SCRAM, as it is the only widely depolyed mechanism with channel binding at the time of writing this document. As already stated in the Business Rules, other mechanisms supporting channel-binding will have some sort of equivalent for "y" and "n".
In SCRAM (RFC 5802 [14]) the client-first-message contains three possible values in the GSS-header part:

[11] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.
[12] RFC 5802: Salted Challenge Response Authentication Mechanism (SCRAM) SASL and GSS-API Mechanisms <http://tools.ietf.org/html/rfc5802>.
[13] XEP-0474: SASL SCRAM Downgrade Protection <https://xmpp.org/extensions/xep-0474.html>.
[14] RFC 5802: Salted Challenge Response Authentication Mechanism (SCRAM) SASL and GSS-API Mechanisms <http://tools.ietf.org/html/rfc5802>.

1. y - The client would have used channel-binding, but the server did not offer any.
2. n - The client does not support channel-binding (even if the server offered any).
3. p=<cb-name> - The name of the channel-binding, the client wants to use.

The RFC explains, that sending ”y” when the server advertised channel-binding support is to be used as a MITM-detection. An attacker stripping out all *-PLUS variants can be detected this way: the server knows it advertised them and the client reports via ”y”, that it would have used them if it saw them advertised, so the server can abort the authentication in this case.
This works reasonably well, if there is only one single channel-binding possible. But that’s the exact assumption this specification now changes: it is now possible to advertise a list of different channel-bindings for the client to choose.
But this creates a problem: what to do if the server advertised a list of channel-binding algorithms, but the client doesn’t support any of these? Assuming there is no MITM attacker present, the client can’t send ”y” in the GSS-header, because the server would then abort the authentication because it advertised channel-bindings. Sending ”n” and continuing without channel-binding is fine in this case, but it won’t be if a MITM attacker were present. [15]
Any MITM attacker could just manipulate the list of channel-bindings advertised using this specification to just list some dummy mechanisms the client doesn’t support. If the client acted like in the non-attacker case depicted in the last paragraph and sent ”n”, the attacker would have successfully downgraded the client to non-channel-binding. The client won’t be able to distinguish the attacker case from the non-attacker one.
The rules in the Business Rules section fix that loophole. Tls-server-end-point was picked, because it is the lowest denominator that can be implemented by virtually everyone and even though it isn’t as strong as tls-exporter or tls-unique, it still catches many attacks. **This only works reliably, if *every* server supports and announces tls-server-end-point.**
A client following the rules above and seeing a server-advertised list without tls-sever-end-point, immediately knows, that some attacker tampered with the list of channel-binding types and can abort the authentication. It never needs to send ”n” even though it supports channel-binding and can still safely send ”y” if it doesn’t see any ’*-PLUS’ variants and channel-bindings announced.

5 IANA Considerations

This document requires no interaction with the Internet Assigned Numbers Authority (IANA) [16].

[15] This isn’t fully hypothetical. An older client might only support tls-unique, while the server only advertises tls-exporter (which can be used for tls 1.2, too, if the extended master secret is used). Blocking the connection entirely is of course at the discretion of the client developer, but it hinders interoperability, especially while phasing out one channel-binding-type and introducing a new one.

[16] 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/>.

6 XMPP Registrar Considerations

Add the 'urn:xmpp:sasl-cb:0' namespace to the registry:

```
<var>
  <name>urn:xmpp:sasl-cb:0</name>
  <desc>Expose supported channel-binding types to clients</desc>
  <doc>XEP-0440</doc>
</var>
```

7 XML Schema

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

<xs:schema
    xmlns:xs='http://www.w3.org/2001/XMLSchema'
    xmlns='urn:xmpp:sasl-cb:0'
    targetNamespace='urn:xmpp:sasl-cb:0'
    elementFormDefault='qualified'>

  <xs:annotation>
    <xs:documentation>
      The protocol documented by this schema is defined in
      XEP-0440: https://xmpp.org/extensions/xep-0440.html
    </xs:documentation>
  </xs:annotation>

  <xs:element name='sasl-channel-binding'>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref='channel-binding' maxOccurs='unbounded'/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name='channel-binding'>
    <xs:complexType>
      <xs:attribute name='type' type='xs:string'/>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

8 Acknowledgements

Thanks to Sam Whited for the discussion about the underlying issue and incentivizing me to come up with this extension. Further thanks goes to Ruslan N. Marchenko for pointing out the possible MITM attack vector. Last but not least, Dave Cridland, Thilo Molitor, and Simon Josefsson provided valuable feedback.