XEP-0474: SASL SCRAM Downgrade Protection

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

2023-12-04
Version 0.3.0

Status	Type	Short Name
Experimental	Standards Track	SSDP

This specification provides a way to secure the SASL and SASL2 handshakes against method and channel-binding downgrades.

Legal

Contents

1 Introduction

RFC 6120 [1] and Extensible SASL Profile (XEP-0388) [2] define a way to negotiate SASL mechanisms. When used together with SCRAM mechanisms (RFC 5802 [3]) and channel-binding (SASL Channel-Binding Type Capability (XEP-0440) [4]) the mechanism selection is protected against downgrade attacks by an active MITM tampering with the TLS channel and advertised SASL mechanisms. Yet, the negotiation of the channel-binding types is not protected against such downgrade attacks.
SASL Channel-Binding Type Capability (XEP-0440) [5] tries to mitigate this by making the "tls-server-end-point" (RFC 5929 [6]) channel-binding mandatory to implement for servers. But that leaves clients not able to implement this type, or any channel-binding at all, vulnerable to downgrades of channel-binding types and SASL mechanisms. Furthermore "tls-server-end-point" provides weaker security guarantees than other channel-bindings like for example "tls-exporter" (defined in RFC 5705 [7] and RFC 9266 [8]).
Most clients use pinning of channel-binding types and SASL mechanisms to protect against downgrade attacks, but this protection is incomplete. First of all this can not protect the first connection. Second server operators can not deactivate previously advertised mechanisms (clients having pinned that mechanism will not authenticate anymore). This can be used by attackers to trick users into reinstalling/reconfiguring their chat app to MITM the then first connection (which again is not protected by pinning).
This specification aims to solve these issues by spcifying a downgrade protection for both SASL mechanisms and channel-binding types using an optional SCRAM attribute (see RFC 5802 [9]). This specification can be used for SASL1 (RFC 6120 [10]) and SASL2 (Extensible SASL Profile (XEP-0388) [11]) profiles as well as any other SASL profile.
Note: In the long term the author strives to publish this as an RFC rather than a XEP to also make this protection available to other protocols, after gaining implementation experience.

[1] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.
[2] XEP-0388: Extensible SASL Profile <https://xmpp.org/extensions/xep-0388.html>.
[3] RFC 5802: Salted Challenge Response Authentication Mechanism (SCRAM) SASL and GSS-API Mechanisms <http://tools.ietf.org/html/rfc5802>.
[4] XEP-0440: SASL Channel-Binding Type Capability <https://xmpp.org/extensions/xep-0440.html>.
[5] XEP-0440: SASL Channel-Binding Type Capability <https://xmpp.org/extensions/xep-0440.html>.
[6] RFC 5929: Channel Bindings for TLS <http://tools.ietf.org/html/rfc5929>.
[7] RFC 5705: Keying Material Exporters for Transport Layer Security (TLS) <http://tools.ietf.org/html/rfc5705>.
[8] RFC 9266: Channel Bindings for TLS 1.3 <http://tools.ietf.org/html/rfc9266>.
[9] RFC 5802: Salted Challenge Response Authentication Mechanism (SCRAM) SASL and GSS-API Mechanisms <http://tools.ietf.org/html/rfc5802>.
[10] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.
[11] XEP-0388: Extensible SASL Profile <https://xmpp.org/extensions/xep-0388.html>.

2 Glossary

This specification uses some abbreviations:

- MITM: man-in-the-middle
- CA: Certificate Authority
- SASL1: the XMPP SASL profile specified in RFC 6120 [12]
- SASL2: the XMPP SASL profile specified in Extensible SASL Profile (XEP-0388) [13]

3 Requirements

This protocol was designed with the following requirements in mind:

- Allow detection of SASL mechanism downgrades even if no channel-binding is in use.
- Allow detection of downgrades of channel-binding types.
- Support all currently defined and future SCRAM mechanisms (RFC 5802 [14] and RFC 7677 [15]).
- Allow for (more) protocol agility compared to pinning.
- Be not less secure than pinning when using the SCRAM family of mechanisms (or some similar challenge-response based authentication mechanism).

Note that this specification intentionally leaves out support for SASL PLAIN. If server and client support PLAIN, no protection against SASL method or channel-binding downgrades is possible and the security relies solely on the underlying TLS channel. As explained in § 13.8.3 of RFC 6120 [16], servers and clients SHOULD NOT support SASL PLAIN unless it is required by the authentication backend.
Instead of pinning a concrete SASL mechanism it might be an acceptable approach to only pin if the server previously supported at least one mechanism better than SASL-PLAIN. This would ensure that the authentication won't fall back to SASL-PLAIN in the future, but also won't hinder protocol agility for the SCRAM family of SASL mechanisms etc..

[12] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.
[13] XEP-0388: Extensible SASL Profile <https://xmpp.org/extensions/xep-0388.html>.
[14] RFC 5802: Salted Challenge Response Authentication Mechanism (SCRAM) SASL and GSS-API Mechanisms <http://tools.ietf.org/html/rfc5802>.
[15] RFC 7677: SCRAM-SHA-256 and SCRAM-SHA-256-PLUS Simple Authentication and Security Layer (SASL) Mechanisms <http://tools.ietf.org/html/rfc7677>.
[16] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.

4 Protection Scenarios

In the following, the limitations of pinning are shown and explained how these can be solved with this specification. This list is by no means meant to be exhaustive. See also Security Considerations for a more complete attack model and problem description.

4.1 MITM Downgrades Channel-Binding Method

The attacker is bit more sophisticated and able to get the private key for the server's certificate and they are only targeting a special user (or a small group of users). This user already connected to the server and thus pinned tls-exporter and tls-unique channel bindings to be supported by the server and thus refuses to downgrade to tls-server-end-point (or to no channel-binding at all).
The attacker now tries to downgrade to tls-server-end-point nonetheless and the client will, thanks to pinning, detect this downgrade, alert the user and refuse to connect.
But no matter how scary this alert message is, most users will first of all think of a bug, delete their account in the client and set it up again. Especially if they ask friends that don't have this connection problems (and may even tell them to reinstall the app, too).
After reinstalling the app and setting up their account again, everything works as before and the client even shows channel-binding in use (if the user even checks that), but in fact now the MITM attacker was successful and is able to intercept or even manipulate every stanza sent/received on the user's connection.
But if the client and server in this scenario supported this specification, the MITM would not have been successful. No matter how often the user tries to reconfigure/reinstall their client, the attacker won't succeed in MITMing the connection of that user.

4.2 Protocol Agility: SASL Methods

In this scenario, we have a setup where either the client or server are not able to do channel-binding. That may well be due to restrictions of the client platform, for example a web based client using BOSH or Websockets, or the server software/tls implementation.
Now the server operator briefly activates a SASL method. Let's say they activate SCRAM-SHA-512 and previously had a working setup that only advertised and supported SCRAM-SHA-1. But after activating that, they become aware that this setup has some bugs: some clients are able to authenticate using SCRAM-SHA-512 and some don't but still try, resulting in errors yelled at those users.
That may be because the server software has bugs (those bugs may persist in deployments for a very long time), maybe some clients have bugs in their SCRAM-SHA-512 implementation (but not in their SCRAM-SHA-1 implementation), or maybe the server operator simply made some configuration errors they are not able to fix (quickly).
Just going back to the old known good configuration will solve these issues for those clients not being able to properly use SCRAM-SHA-512, but since all clients that **were** able to authen-

ticate using SCRAM-SHA-512 now pinned that one, disabling this new SASL method again, will make those clients not connect anymore. That's more or less a DOS vector introduced by pinning.
If this specification was used, the clients would not need to pin anything, because the SSDP hash included in the SCRAM handshake will detect that this apparent downgrade is in fact no real downgrade but only a legitimate server configuration change.
See Attack model 2 for an attack that involves a platform not supporting channel-binding that's also mitigated by this specification. More than that: web clients not being able to permanently store any pinning information would still be ("stateless") protected by this specification.

4.3 Protocol Agility: Channel-Binding Types

A server is configured to use strong channel-binding (tls-exporter / tls-unique) but since the userbase grew the server operator decides to do TLS offloading and thus can only offer tls-server-end-point channel-binding (which is still way better than no channel-binding at all).
Another reason to go from tls-exporter / tls-unique to tls-server-end-point may well be a bug in the server/tls library. And a slight modification of this scenario would be the server operator disabling channel-binding altogether (for the same reasons).
But since clients are pinning channel-binding types, this configuration change, albeit legitimate, will be erroneously detected as attack and clients won't connect to the server anymore.
If this specification was used, the clients would have been able to distinguish that legitimate configuration change from an attack and not drop the connection.

5 Attack Model

In the following sections, different attack models will be discussed.

5.1 Attack Model 1 (List of Channel-Binding Types)

Scenario: Bob connects to Alice's XMPP server using a client of his choice supporting SCRAM and channel-binding, Eve wants to MITM this connection. Neither Alice's server nor Bob's client support SASL PLAIN, but only the SCRAM family of SASL mechanisms.
Prerequisites: Eve, the MITM attacker, managed to either steal the cert+key of Alice's XMPP server or to convince some CA to give out a cert+key for Alice's XMPP domain. Maybe Bob even installed a CA of his employer/school and now gets MITMed by his employer/school.
Given this scenario and prerequisites, Eve now can passively MITM the XMPP connection, but Bob and Alice are using channel-binding and this allows them to detect Eve and abort authentication. This forces Eve to be an active attacker, manipulating the data in the XMPP stream to get rid of the channel-binding. Eve does so by changing the list of server-advertised

channel-bindings to only include some (fictional) channel-binding types she is sure the client does not support. Bob's client now has the following choices (see also the Security Considerations of SASL Channel-Binding Type Capability (XEP-0440) [17]):

1. Authenticate without using channel-binding and signal to the server that the client *does not* support channel-binding (”n” GS2-flag)
2. Authenticate without using channel-binding and signal to the server that the client *does* support channel-binding (”y” GS2-flag)
3. Try to authenticate using *some* channel-binding type
4. Try to authenticate using the *pinned* channel-binding type
5. Fall back to use the lowest denominator: ”tls-server-end-point”

Case 1 is a successful downgrade from channel-binding to non-channel-binding authentication, Eve ”wins”.
Case 2 will always fail the authentication if the server supports channel-binding, Eve does not ”win”. But authentication will fail even if there is no MITM present but server and client simply happen to have no mutually supported channel-binding type.
Case 3 can result in a successful or failed authentication, depending on wether the server supports the type randomly selected by the client. Unfortunately a failed authentication due to selecting the wrong channel-binding type can not be distinguished from a failed authentication because of invalid credentials etc. Thus authentication using *some* channel-binding type will slow down authentication speed, because the client has to cycle through all channel-binding types it supports until it finds one the server supports (and eventually fall back to no channel-binding, if all channel-binding types have been tried). So, if server and client have mutually supported channel-binding types, Eve won't ”win”, but authentication will potentially need many roundtrips. If they don't have mutually supported channel-bindig types, Eve wouldn't have had to manipulate the channel-binding list in the first place.
Case 4 does not help on first authentication. This could be neglected, but since channel-binding types aren't that easily ordered by percieved strength and could legitimately change, this could effectively lead to a Denial of Service. For example Alice might want to offload TLS termination because of higher server load and now her server does not support ”tls-exporter” anymore but only ”tls-server-end-point”. A client pinning ”tls-exporter” would not be able to connect to Alice's server anymore after the TLS offloading is in place.
Case 5 won't help if Eve managed to steal the cert+key (or the server either somehow does not support the ”tls-server-end-point” type).
This specification solves the problems outlined above by adding an optional SCRAM attribute containing the hash of the client-perceived list of channel-binding types that can be checked by the server and will be cryptographically signed by the authentication password used for SCRAM.

[17] XEP-0440: SASL Channel-Binding Type Capability <https://xmpp.org/extensions/xep-0440.html>.

5.2 Attack Model 2 (SASL Mechanism List)

Scenario: Bob connects to Alice's XMPP server using a client of his choice supporting SCRAM but **no** channel-binding, Eve wants to MITM this connection. Neither Alice's server nor Bob's client support SASL PLAIN, but only the SCRAM family of SASL mechanisms. Eve wants to downgrade the used SCRAM mechanism to something weak that she is able to break in X hours/days (For example some time in the future SCRAM-SHA-1 might be broken that way and the underlying password could be recovered investing X hours/days of computing time. But SCRAM-SHA-1 might still be supported by servers for backwards compatibility with older clients only supporting SCRAM-SHA-1 but not SCRAM-SHA-256 etc.).
Prerequisites: Eve, the MITM attacker, managed to either steal the cert+key of Alice's XMPP server or to convince some CA to give out a cert+key for Alice's XMPP domain. Maybe Bob even installed a CA of his employer/school and now gets MITMed by his employer/school.
Given this scenario and prerequisites, Eve now can passively MITM the XMPP connection, but if Eve wants to actively downgrade the SASL mechanism used by Bob, he has to actively change the server-advertised SASL mechanism list. In this scenario Eve actively removes all SCRAM mechanisms but SCRAM-SHA-1 from the server-advertised list to force Bob's client to use SCRAM-SHA-1. Neither Alice nor Bob would detect that.
Pinning of SASL mechanisms could be used for that, but in doing this, Alice would loose some flexibility. She might have briefly activated SCRAM-SHA-512 and deactivated it again. Now Bob's client can not authenticate using SCRAM-SHA-512 anymore and authentication will always fail, if pinning is used. Pinning won't help on first connection either. See above for a pinning + SSDP compromise when still supporting SASL PLAIN.
This specification solves this problem by adding an optional SCRAM attribute containing the hash of the client-perceived SASL mechanism list that can be checked by the server and will be cryptographically signed by the authentication password used for SCRAM.

6 Protocol

Sections 5.1 and 7 of RFC 5802 [18] allow for arbitrary optional attributes inside SCRAM messages. This specification uses those optional attribute to implement a downgrade protection.

6.1 Server Sends Downgrade Protection Hash

The server calculates a hash of the list of SASL mechanisms and channel-binding types it advertised as follows.
Note: All sorting operations MUST be performed using "i;octet" collation as specified in Section 9.3 of RFC 4790 [19].

[18]RFC 5802: Salted Challenge Response Authentication Mechanism (SCRAM) SASL and GSS-API Mechanisms <http://tools.ietf.org/html/rfc5802>.
[19]RFC 4790: Internet Application Protocol Collation Registry <http://tools.ietf.org/html/rfc4790>.

1. Initialize an empty ASCII string S
2. Sort all server-advertised SASL mechanisms and append them to string S joined by delimiter ”,” (%x2C)
3. If the server used SASL Channel-Binding Type Capability (XEP-0440) [20] to advertise channel-bindings, append ”|” (%x7C) to S
4. If the server used SASL Channel-Binding Type Capability (XEP-0440) [21] to advertise channel-bindings, sort all server-advertised channel-binding types and append them to string S joined by delimiter ”,” (%x2C)
5. Hash S using the same hash mechanism as used for the SCRAM mechanism currently in use and encode the result using base64

The server then adds the optional attribute ”d” with the base64 encoded hash obtained in step 5 to its server-first-message.
Note: If the server simultaneously advertises SASL1 and SASL2, only the mechanism list of the SASL protocol the client uses for authentication MUST be considered for hashing.

6.2 Client Verifies The Downgrade Protection Hash

Upon receiving the server-first-message the client calculates its own base64 encoded hash using the list of SASL mechanisms and channel-binding types the server advertised using SASL1 or SASL2 and SASL Channel-Binding Type Capability (XEP-0440) [22] by applying the same algorithm as defined in Server Sends Downgrade Protection Hash.
The client then extracts the base64 encoded hash presented by the server in the optional attribute ”d” and compares it to its own hash. If the hashes match, the list of SASL mechanisms and channel-binding types has not been changed by an active MITM.
If the hashes do not match, the client MUST fail the authentication. It MAY additionally show a user-facing warning message about an active MITM. If the hashes match, an attacker could still have manipulated them. If so, the server will always fail the authentication according to RFC 5802 [23] because the client-proof will not be based upon the correct SSDP value.

6.3 Full Example

This sections contains an example based on the ones provided in Extensible SASL Profile (XEP-0388) [24].

[20] XEP-0440: SASL Channel-Binding Type Capability <https://xmpp.org/extensions/xep-0440.html>.
[21] XEP-0440: SASL Channel-Binding Type Capability <https://xmpp.org/extensions/xep-0440.html>.
[22] XEP-0440: SASL Channel-Binding Type Capability <https://xmpp.org/extensions/xep-0440.html>.
[23] RFC 5802: Salted Challenge Response Authentication Mechanism (SCRAM) SASL and GSS-API Mechanisms <http://tools.ietf.org/html/rfc5802>.
[24] XEP-0388: Extensible SASL Profile <https://xmpp.org/extensions/xep-0388.html>.

Listing 1: Full SCRAM-SHA-1-PLUS authentication flow using the optional attribute defined in this spec

```
<!-{}-
  Client sending stream header
-{}->
<stream:stream
  from='user@example.org'
  to='example.org'
  version='1.0'
  xml:lang='en'
  xmlns='jabber:client'
  xmlns:stream='http://etherx.jabber.org/streams'>

<!-{}-
  Server responding with stream header and features
-{}->
<stream:stream
  from='example.org'
  id='++TR84Sm6A3hnt3Q065SnAbbk3Y='
  to='user@example.org'
  version='1.0'
  xml:lang='en'
  xmlns='jabber:client'
  xmlns:stream='http://etherx.jabber.org/streams'>
<stream:features>
  <authentication xmlns='urn:xmpp:sasl:2'>
    <mechanism>SCRAM-SHA-1</mechanism>
    <mechanism>SCRAM-SHA-1-PLUS</mechanism>
    <inline xmlns='urn:xmpp:sasl:2'>
      <!-{}- Server indicates that XEP-0198 can be negotiated "inline"
          -{}->
      <enable xmlns='urn:xmpp:sm:3'/>
      <!-{}- Server indicates support for XEP-0386 Bind 2 -{}->
      <bind xmlns='urn:xmpp:bind2:1'/>
    </inline>
  </authentication>
  <!-{}- Channel-binding information provided by XEP-0440 -{}->
  <sasl-channel-binding xmlns='urn:xmpp:sasl-cb:0'>
    <channel-binding type='tls-server-end-point'/>
    <channel-binding type='tls-exporter'/>
  </sasl-channel-binding>
</stream:features>

<!-{}-
  Client initiates authentication using SCRAM-SHA-1-PLUS and channel-
      binding type "tls-exporter"
-{}->
<authenticate xmlns='urn:xmpp:sasl:2' mechanism='SCRAM-SHA-1-PLUS'>
  <!-{}- Base64 of: 'p=tls-exporter,,n=user,r=12C4CD5C-E38E-4A98-8F6D
```

```
      -15C38F51CCC6' -{}->
  <initial-response>
      cD10bHMtZXhwb3J0ZXIsLG49dXNlcixyPTEyQzRDRDVDLUUzOEUtNEE5OC04RjZELTE1QzM4RjUxQ0ND
      ==</initial-response>
  <user-agent id='d4565fa7-4d72-4749-b3d3-740edbf87770'>
    <software>AwesomeXMPP</software>
    <device>Kiva's␣Phone</device>
␣␣</user-agent>
</authenticate>

<!-{}-
␣␣SCRAM-SHA-1-PLUS␣challenge␣issued␣by␣the␣server␣as␣defined␣in␣RFC␣
    5802
␣␣including␣the␣base64␣encoded␣SHA-1␣hash␣of␣the␣mechanism␣and␣channel
    -binding␣lists.
␣␣Attribute␣"d"␣contains␣base64␣encoded␣SHA-1␣hash␣of␣'SCRAM-SHA-1,
    SCRAM-SHA-1-PLUS|tls-exporter,tls-server-end-point'
␣␣Base64␣of:␣'r=12C4CD5C-E38E-4A98-8F6D-15C38F51CCC6a09117a6-ac50-4f2f
    -93f1-93799c2bddf6,s=QSXCR+Q6sek8bf92,i=4096,d=
    dRc3RenuSY9ypgPpERowoaySQZY='
-{}->
<challenge␣xmlns='urn:xmpp:sasl:2'>
␣␣
    cj0xMkM0Q0Q1Qy1FMzhFLTRBOTgtOEY2RC0xNUMzOEY1MUNDQzZhMDkxMTdhNi1hYzUwLTRmMmYtOTNmMS
    =
</challenge>

<!-{}-
␣␣The␣client␣responds␣with␣the␣base64␣encoded␣SCRAM-SHA-1-PLUS␣client-
    final-message␣(password:␣'pencil')
␣␣The␣c-attribute␣contains␣the␣GS2-header␣and␣channel-binding␣data␣
    blob␣as␣defined␣in␣RFC␣5802.
␣␣Base64␣of:␣'c=cD10bHMtZXhwb3J0ZXIsLFRISVMgSVMgRkFLRSBDQiBEQVRB,r=12
    C4CD5C-E38E-4A98-8F6D-15C38F51CCC6a09117a6-ac50-4f2f-93f1-93799
    c2bddf6,p=YrZgr+FXrBmtcPY6weDLAFcSb9k='
-{}->
<response␣xmlns='urn:xmpp:sasl:2'>
␣␣
    Yz1jRDEwYkhNdFpYaHdiM0owWlhJc0xGUklTVk1nU1ZNZ1JrRkxSU0JEUWlCRVFWUkIscj0xMkM0Q0Q1Qy

</response>

<!-{}-
␣␣The␣server␣accepted␣this␣authentication,␣no␣tampering␣with␣the␣
    advertised␣SASL␣mechanisms␣or␣channel-bindings␣was␣detected.
-{}->
<success␣xmlns='urn:xmpp:sasl:2'>
␣␣<!-{}-␣Base64␣of:␣'v=bWt5Od0DkLlIvhb4BDO8kzkx0LM='␣-{}->
␣␣<additional-data>
```

```
    dj1iV3Q1T2QwRGtMbEl2aGI0QkRPOGt6a3gwTE09
  </additional-data>
  <authorization-identifier>user@example.org</authorization-identifier
    >
</success>
```

7 Security Considerations

Using SCRAM attributes makes them part of the HMAC signatures used in the SCRAM protocol flow efficiently protecting them against any MITM attacker not knowing the password used.

8 IETF Interaction

This protocol shall be superseded by any IETF RFC providing some or all of the functionality provided by this specification. If such a specification exists implementations SHOULD NOT implement this XEP and SHOULD implement the superseding RFC instead.

9 IANA Considerations

This document requires no interaction with the Internet Assigned Numbers Authority (IANA) [25].

10 XMPP Registrar Considerations

This specification does not need any interaction with the XMPP Registrar [26].

11 XML Schema

This specification does not specify any new XML elements.

[25] 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/>.

[26] 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/>.