XMPP SASL2

Kitten WG Virtual Meeting

Matthew Wild / Thilo Molitor

2023-01-17

Section 1

Background

SASL in XMPP

- Specified in RFC 3920, RFC 6120
- Base64 encoded, within XML
- Universal support in implementations
- Mostly SCRAM-SHA-1 in deployments, some PLAIN

Section 2

SASL2

Status

- First implementation and draft in 2017 by Dave Cridland as XEP-0388: "Extensible SASL Profile"
- Quiet for ~5 years
- Implemented and revised in 2022 by 3+ projects
- Foundation for "FAST", "Bind 2", and other enhancements

Changes from RFC 6120

- Secondary tasks (for multi-factor, upgrades, . . .)
- Stable client identifiers
- Channel binding negotiation

Authentication flow

- `<authenticate>` (select mechanism, send any initial data)
- `<challenge>`
- `<response>`
- `<success>` or `<failure>`

SASL2 Tasks

- `<success>` or `<failure>` **or** `<continue>`
- `<next>` (execute a task, optionally with initial data)
- `<challenge>`
- `<response>`
- `<success>` or `<failure>` or `<continue>` (more tasks required)

Multi-factor auth

Simplest flow:

- Server issues a challenge - “provide OTP”
- Client provides OTP

Other challenge types are more complex:

- Different kinds of OTP (TOTP, SMS codes, . . .)
- Out-of-band challenges (click link in email, push notifications, . . .)
- FIDO2. . .

Task flow

Changed from strict challenge-response: `<challenge/>` -> `<response/>`

To bidirectional task-defined steps: `<task-data/>` <-> `<task-data/>`

More complex tasks (including some multi-factor types) demand structured data. XML is a great fit for this in XMPP.

Client identifiers

- User-Agent for authentication (diagnostics)
- Stable unique identifier for returning clients
  - Assists with certain mechanisms, such as hashed tokens
  - authcid, but client level

Upgrade Tasks

- Servers often only store salted hashes for SCRAM-SHA-1
- Upgrade tasks can be used to add the salted hash for SCRAM-SHA-256 to server storage
- Implemented as SASL2 tasks:
  - Before SASL, server advertises what upgrades are possible (e.g. “we support SCRAM-SHA-256, but have no compatible credentials stored for you yet”)
  - Client tells the server what mechanisms to upgrade to
  - Server issues the requested tasks providing the needed data (salt, iteration count etc.)
  - Client sends the needed data for each task (usually a new salted hash)

Upgrade example: SCRAM-SHA-1 to SCRAM-SHA-256

Initial authentication request from client:

```
<authenticate xmlns='urn:xmpp:sasl:2'
              mechanism='SCRAM-SHA-1-PLUS'>
  <upgrade>UPGR-SCRAM-SHA-256</upgrade>
  <initial-response>[...]</initial-response>
</authenticate>
```

Upgrade example: Auth mechanism complete

After the SCRAM exchange, the server returns <continue> (tasks) instead of <success>.

```
<continue xmlns='urn:xmpp:sasl:2'>
  <additional-data>
    [...]
  </additional-data>
  <tasks>
    <task>UPGR-SCRAM-SHA-256</task>
  </tasks>
</continue>
```

Upgrade example: Client initiates task

```
<next xmlns='urn:xmpp:sasl:2' task='UPGR-SCRAM-SHA-256'/>
```

Upgrade example: Server provides parameters

The server sends the required salt and iteration count.

```
<task-data xmlns='urn:xmpp:sasl:2'>
  <salt
        xmlns='urn:xmpp:scram-upgrade:0'
        iterations='4096'>
    [...]
  </salt>
</task-data>
```

Upgrade example: Client provides credentials

The client responds with the base64 encoded SaltedPassword.

```
<task-data xmlns='urn:xmpp:sasl:2'>
  <hash xmlns='urn:xmpp:scram-upgrade:0'>
    BzOnw3P[...]c5H4bOLlPZ=
  </hash>
</task-data>
```

Upgrade example: Authentication complete

Finally, the server sends a success after adding the salted SHA-256 hash to its database.

```
<success xmlns='urn:xmpp:sasl:2'>
  <authorization-identifier>
      user@example.org
  </authorization-identifier>
</success>
```

Section 3

Channel-binding negotiation

Channel-binding negotiation

- Multiple channel-binding (CB) types can be used
  - tls-exporter, tls-server-end-point, tls-unique
- With SCRAM (and OPAQUE?) client does not know what's supported by the server
- Using an unsupported CB type leads to failed authentication
  - No way for the client to know why (and retry with another type)
  - Blindly falling back to non-CB would circumvent MITM protection of CB
  - In any case: retrying authentication will slow down the overall authentication

Channel-binding downgrades #1

- If we assume a MITM in our TLS channel, CB can prevent authentication
- But: what to do if the attacker manipulates the server advertised CB types?
  - If no known type is listed, client could fall back to non-CB auth
  - That would be a successful downgrade
  - (Downgrades from tls-exporter to tls-server-end-point are also possible)

Channel-binding downgrades #2

- XEP-0474 provides a way to add a hash of the CB list to the SCRAM/OPAQUE handshake (attribute)
  - This cryptographically signs the CB list with the password used for authentication
  - Server can detect mismatch between advertised CB list and client perceived list (fail authentication)

Channel-binding downgrades #3

- XEP-0474 adds support for downgrade detection of SCRAM/OPAQUE mechanisms, too (even without CB)
  - Using the hash of the mechanisms list
  - A MITM that could break SCRAM-SHA-1 in X days
  - MITM tries to downgrade from SCRAM-SHA-256 to SCRAM-SHA-1
  - This can be detected and the user alerted to change their password

Section 4

Conclusion

Further reading

Contact:

- Matthew Wild (mailto/xmpp: me@matthewild.co.uk)
- Thilo Molitor (mailto:thilo+xmpp@eightysoft.de, xmpp:thilo.molitor@juforum.de)

Specifications:

- XEP-0388: Extensible SASL Profile (aka “SASL2”)
- XEP-0474: SASL SCRAM Downgrade Protection
- XEP-xxxx: Fast Authentication Streamlining Tokens