Avro SASL profile

Table of contents

1. Introduction

SASL (RFC 2222) provides a framework for authentication and security of network protocols. Each protocol that uses SASL is meant to define a SASL *profile*. This document provides a SASL profile for connection-based Avro RPC.

2. Overview

SASL negotiation proceeds as a series of message interactions over a connection between a client and server using a selected SASL *mechanism*. The client starts this negotiation by sending its chosen mechanism name with an initial (possibly empty) message. Negotiation proceeds with the exchange of messages until either side indicates success or failure. The content of the messages is mechanism-specific. If the negotiation succeeds, then the session can proceed over the connection, otherwise it must be abandoned.

Some mechanisms continue to process session data after negotiation (e.g., encrypting it), while some specify that further session data is transmitted unmodifed.

3. Negotiation

3.1. Commands

Avro SASL negotiation uses four one-byte commands.

- `0: START` Used in a client's initial message.
- `1: CONTINUE` Used while negotiation is ongoing.
- `2: FAIL` Terminates negotiation unsuccessfully.
- `3: COMPLETE` Terminates negotiation sucessfully.

The format of a START message is:

```
| 0 | 4-byte mechanism name length | mechanism name | 4-byte payload length
| payload data |
```

The format of a CONTINUE message is:

```
| 1 | 4-byte payload length | payload data |
```

The format of a FAIL message is:

```
| 2 | 4-byte message length | UTF-8 message |
```

The format of a COMPLETE message is:

```
| 3 | 4-byte payload length | payload data |
```

3.2. Process

Negotiation is initiated by a client sending a START command containing the client's chosen mechanism name and any mechanism-specific payload data.

The server and client then interchange some number (possibly zero) of CONTINUE messages. Each message contains payload data that is processed by the security mechanism to generate the next message.

Once either the client or server send a FAIL message then negotiation has failed. UTF-8-encoded text is included in the failure message. Once either a FAIL message has been sent or recieved, or any other error occurs in the negotiation, further communication on this connection must cease.

Once either the client or server send a COMPLETE message then negotiation has completed successfully. Session data may now be transmitted over the connection until it is closed by either side.

4. Session Data

If no SASL QOP (quality of protection) is negotiated, then all subsequent writes to/reads over this connection are written/read unmodified. In particular, messages use Avro framing, and are of the form:

```
| 4-byte frame length | frame data | ... | 4 zero bytes |
```

If a SASL QOP is negotiated, then it must be used by the connection for all subsequent messages. This is done by wrapping each non-empty frame written using the security mechanism and unwrapping each non-empty frame read. The length written in each non-empty frame is the length of the wrapped data. Complete frames must be passed to the security mechanism for unwrapping. Unwrapped data is then passed to the application as the content of the frame.

If at any point processing fails due to wrapping, unwrapping or framing errors, then all further communication on this connection must cease.

5. Anonymous Mechanism

The SASL anonymous mechanism (RFC 2245) is quite simple to implement. In particular, an initial anonymous request may be prefixed by the following static sequence:

```
| 0 | 0009 | ANONYMOUS | 0000 |
```

If a server uses the anonymous mechanism, it should check that the mechanism name in the start message prefixing the first request recieved is 'ANONYMOUS', then simply prefix its initial response with a COMPLETE message of:

```
| 3 | 0000 |
```

If an anonymous server recieves some other mechanism name, then it may respond with a FAIL message as simple as:

```
| 2 | 0000 |
```

Note that the anonymous mechanism need add no additional round-trip messages between client and server. The START message can be piggybacked on the initial request and the COMPLETE or FAIL message can be piggybacked on the initial response.