smtpproto

Release 1.1.0.post1

Alex Grönholm

Apr 15, 2021

CONTENTS

This library contains a (client-side) sans-io implementation of the ESMTP protocol. A concrete, asynchronous I/O implementation is also provided, via the AnyIO library.

The following SMTP extensions are supported:

- 8BITMIME
- AUTH
- SIZE (max message size reporting only)
- SMTPUTF8
- STARTTLS

You can find the documentation here.

CHAPTER

ONE

TABLE OF CONTENTS

1.1 Developing new I/O implementations

The procedure to using the SMTP client protocol state machine to communicate with an SMTP server is as follows:

1. Create the state machine (*SMTPClientProtocol*)
2. Connect to the SMTP server using your chosen I/O backend

Sending commands and receiving responses:

1. Call the appropriate method on the state machine
2. Retrieve the outgoing data with *get_outgoing_data()*
3. Use your I/O backend to send that data to the server
4. Use your I/O backend to receive the response data
5. Feed the response data to the state machine using *feed_bytes()*
6. If the return value is an *SMTPResponse* (and not `None`), process the response as appropriate. You can use *is_error()* as a convenience to check if the response code means there was an error.

Establishing a TLS session after connection (optional):

1. Check if the feature is supported by the server (`STARTTLS` is in *extensions*)
2. Send the `STARTTLS` command using *start_tls()*
3. Use your I/O backend to do the TLS handshake in client mode (`wrap_socket()` or whatever you prefer)
4. Proceed with the session as usual

1.2 Developing new authenticators

To add support for a new authentication mechanism, you can create a new class that inherits from either *SMTPAuthenticator* or one of its subclasses. This subclass needs to implement:

- The *mechanism* property
- The *authenticate()* method

The `mechanism` property should return the name of the authentication mechanism (in upper case letters). It is used to send the initial `AUTH` command. If `mechanism` returns `FOOBAR`, the client would send the command `AUTH FOOBAR`.

The `authenticate` method should return an asynchronous generator that yields strings. If the generator yields a nonempty string on the first call, it is added to the `AUTH` command. For example, given the following code, the client would authenticate with the command `AUTH FOOBAR mysecret`:

```
from smtpproto.auth import SMTPAuthenticator

class MyAuthenticator(SMTPAuthenticator):
    @property
    def mechanism(self) -> str:
        return 'FOOBAR'

    async def authenticate(self) -> AsyncGenerator[str, str]:
        yield 'mysecret'
```

For mechanisms such as `LOGIN` that involve more rounds of information exchange, the generator typically yields an empty string first. It will then be sent back the server response text as the `yield` result. The authenticator will then yield its own response, and so forth. See the source code of the *LoginAuthenticator* class for an example.

1.3 Using the concrete I/O implementations

In addition to the sans-io protocol implementation, this library also provides both an asynchronous and a synchronous SMTP client class (*AsyncSMTPClient* and *SyncSMTPClient*, respectively).

Most SMTP servers, however, require some form of authentication. While it would be unfeasible to provide solutions for every possible situation, the examples below should cover some very common cases and should give you a general idea of how to work with SMTP authentication.

For the OAuth2 examples (further below), you need to install a couple dependencies:

- aiohttp
- PyJWT

1.3.1 Sending mail via a local SMTP server

```
from email.message import EmailMessage

import anyio
from smtpproto.auth import PlainAuthenticator
from smtpproto.client import AsyncSMTPClient


async def main():
    async with AsyncSMTPClient(host='localhost', port=25) as client:
        await client.send_message(message)

# If your SMTP server requires basic authentication, this is where you enter that info
authenticator = PlainAuthenticator(username='myuser', password='mypassword')

# The message you want to send
message = EmailMessage()
message['From'] = 'my.name@mydomain.com'
message['To'] = 'somebody@somewhere'
message['Subject'] = 'Test from smtpproto'
message.set_content('This is a test.')
```

(continues on next page)

(continued from previous page)

```
# Actually sends the message by running main()
anyio.run(main)
```

1.3.2 Sending mail via Gmail

The developer documentation for the G Suite describes how to use the XOAUTH2 mechanism for authenticating against the Gmail SMTP server. The following is a practical example of how to extend the *OAuth2Authenticator* class to obtain an access token and use it to send an email via Gmail.

The following example assumes the presence of an existing G Suite service account authorized to send email via SMTP (using the `https://mail.google.com/` scope).

```
from datetime import datetime, timedelta
from email.message import EmailMessage

import aiohttp
import jwt

import anyio
from smtpproto.auth import OAuth2Authenticator
from smtpproto.client import AsyncSMTPClient


class GMailAuthenticator(OAuth2Authenticator):
    def __init__(self, username: str, client_id: str, private_key: str):
        super().__init__(username)
        self.client_id = client_id
        self.private_key = private_key

    async def get_token_async(self):
        webtoken = jwt.encode({
            'iss': self.client_id,
            'scope': 'https://mail.google.com/',
            'aud': 'https://oauth2.googleapis.com/token',
            'exp': datetime.utcnow() + timedelta(minutes=1),
            'iat': datetime.utcnow(),
            'sub': self.username
        }, self.private_key, algorithm='RS256')

        data = {'grant_type': 'urn:ietf:params:oauth:grant-type:jwt-bearer',
                'assertion': webtoken.decode('ascii')}
        async with aiohttp.request('POST', 'https://oauth2.googleapis.com/token',␣
↪data=data,
                                   raise_for_status=True) as response:
            json_body = await response.json()

        return json_body['access_token'], json_body["expires_in"]


async def main():
    async with AsyncSMTPClient(host='smtp.gmail.com', authenticator=authenticator) as␣
↪client:
        await client.send_message(message)
```

(continues on next page)

(continued from previous page)

```
# Your gmail user name
me = 'my.name@gmail.com'

# Service account ID and private key - these have to be obtained from Gmail
client_id = 'yourserviceaccount@yourdomain.iam.gserviceaccount.com'
private_key = '-----BEGIN PRIVATE KEY-----\n...-----END PRIVATE KEY-----\n'
authenticator = GMailAuthenticator(username=me, client_id=client_id, private_
↪key=private_key)

# The message you want to send
message = EmailMessage()
message['From'] = me
message['To'] = 'somebody@somewhere'
message['Subject'] = 'Test from smtpproto'
message.set_content('This is a test.')

# Actually sends the message by running main()
anyio.run(main)
```

1.3.3 Sending mail via Office 365

Warning: It is currently not clear what actual permissions the service account requires. As such, this example *should* work but has never been successfully tested.

The following example assumes the presence of a registered Azure application authorized to send email via SMTP (using the `SMTP.Send` scope). It uses the device code flow to obtain an access token.

In order for the device code flow to work for the registered application, the following settings must be in place:

- The redirect URI for the application must be `https://login.microsoftonline.com/common/oauth2/nativeclient`
- The `Treat application as a public client` option must be enabled
- The `SMTP.Send` permission from `Microsoft Graph` must be added in the configured permissions

In addition, your Azure AD must not have Security defaults enabled.

```
from email.message import EmailMessage

import aiohttp

import anyio
from smtpproto.auth import OAuth2Authenticator
from smtpproto.client import AsyncSMTPClient


class AzureAuthenticator(OAuth2Authenticator):
    def __init__(self, username: str, tenant_id, client_id: str, client_secret: str):
        super().__init__(username)
        self.tenant_id = tenant_id
        self.client_id = client_id
        self.client_secret = client_secret
```

(continues on next page)

(continued from previous page)

```
    async def get_token_async(self):
        data = {'client_id': self.client_id,
                'scope': 'https://outlook.office.com/SMTP.Send',
                'client_secret': self.client_secret,
                'grant_type': 'client_credentials'}
        async with aiohttp.request(
                'POST', f'https://login.microsoftonline.com/{self.tenant_id}/oauth2/
↪v2.0/token',
                data=data, raise_for_status=True) as response:
            json_body = await response.json()

        return json_body['access_token'], json_body["expires_in"]


async def main():
    async with AsyncSMTPClient(host='smtp.office365.com',
                               authenticator=authenticator) as client:
        await client.send_message(message)

# Your Office 365 username/email address
me = 'my.name@office365.com'

# Application (client) ID and secret - these have to be obtained from the Azure portal
tenant_id = '11111111-1111-1111-1111-111111111111'
client_id = '11111111-1111-1111-1111-111111111111'
client_secret = '...'
authenticator = AzureAuthenticator(username=me, tenant_id=tenant_id, client_id=client_
↪id,
                                   client_secret=client_secret)

# The message you want to send
message = EmailMessage()
message['From'] = me
message['To'] = 'somebody@somewhere'
message['Subject'] = 'Test from smtpproto'
message.set_content('This is a test.')

# Actually sends the message by running main()
anyio.run(main)
```

1.4 API Reference

1.4.1 Protocol

class smtpproto.protocol.**ClientState**(*value*)

Bases: enum.Enum

Enumerates all possible protocol states.

greeting_expected = 1

expecting a greeting from the server

greeting_received = 2

received a greeting from the server, ready to authenticate

authenticating = 3
authentication in progress

authenticated = 4
authentication done

ready = 5
ready to send commands

mailtx = 6
in a mail transaction

recipient_sent = 7
sent at least one recipient

send_data = 8
ready to send the message data

data_sent = 9
message data sent

finished = 10
session finished

exception smtpproto.protocol.**SMTPException**
Bases: Exception

Base class for SMTP exceptions.

exception smtpproto.protocol.**SMTPMissingExtension**
Bases: *smtpproto.protocol.SMTPException*

Raised when a required SMTP extension is not present on the server.

exception smtpproto.protocol.**SMTPUnsupportedAuthMechanism**
Bases: *smtpproto.protocol.SMTPException*

Raised when trying to authenticate using a mechanism not supported by the server.

exception smtpproto.protocol.**SMTPProtocolViolation**
Bases: *smtpproto.protocol.SMTPException*

Raised when there has been a violation of the (E)SMTP protocol by either side.

class smtpproto.protocol.**SMTPResponse**(*code*, *message*)
Bases: object

Represents a response from the server.

code: int
response status code (between 100 and 599)

message: str
response message

is_error()
Return True if this is an error response, False if not.

Return type bool

raise_as_exception()
Raise an *SMTPException* from this response.

Return type NoReturn

class smtpproto.protocol.**SMTPClientProtocol**
Bases: object

The (E)SMTP protocol state machine.

property state
The current state of the protocol.

Return type *ClientState*

property needs_incoming_data
`True` if the state machine requires more data, `False` if not.

Return type bool

get_outgoing_data()
Retrieve any bytes to be sent to the server.

Return type bytes

property max_message_size
The maximum size of the email message (in bytes) accepted by the server.

Return type Optional[int]

property auth_mechanisms
The set of authentication mechanisms supported on the server.

Return type FrozenSet[str]

property extensions
The set of extensions advertised by the server.

Return type FrozenSet[str]

authenticate(*mechanism*, *secret=None*)
Authenticate to the server using the given mechanism and an accompanying secret.

Parameters

- **mechanism** (str) – the authentication mechanism (e.g. PLAIN or GSSAPI)
- **secret** (Optional[str]) – an optional string (usually containing the credentials) that is added as an argument to the AUTH XXX command

Return type None

send_authentication_data(*data*)
Send authentication data to the server.

This method can be called when the server responds with a 334 to an AUTH command.

Parameters **data** (str) – authentication data (ASCII compatible; usually base64 encoded)

Return type None

send_greeting(*domain*)
Send the initial greeting (EHLO or HELO).

Parameters **domain** (str) – the required domain name that represents the client side

Return type None

noop()
Send the NOOP command (No Operation).

Return type None

quit()
Send the QUIT command (required to cleanly shut down the session).

Return type None

mail(*sender*, *, *smtputf8=True*)
Send the MAIL FROM command (starts a mail transaction).

Parameters

- **sender** (Union[str, Address]) – the sender's email address
- **smtputf8** (bool) – send the SMTPUTF8 option, if available on the server

Return type None

recipient(*recipient*)
Send the RCPT TO command (declare an intended recipient).

Requires an active mail transaction.

Parameters **recipient** (Union[str, Address]) – the recipient's email address

Return type None

start_data()
Send the DATA command (prepare for sending the email payload).

Requires an active mail transaction, and that at least one recipient has been declared.

Return type None

data(*message*)
Send the actual email payload.

Requires that the DATA command has been sent first.

Parameters **message** (EmailMessage) – the email message

Return type None

reset()
Send the RSET command (cancel the active mail transaction).

Return type None

start_tls()
Send the STARTTLS command (signal the server to initiate a TLS handshake).

Return type None

feed_bytes(*data*)
Feed received bytes from the transport into the state machine.

if this method raises *SMTPProtocolViolation*, the state machine is transitioned to the finished state, and the connection should be closed.

Parameters **data** (bytes) – received bytes

Return type Optional[*SMTPResponse*]

Returns a response object if a complete response was received, None otherwise

Raises ***SMTPProtocolViolation*** – if the server sent an invalid response

1.4.2 Authentication

class smtpproto.auth.**LoginAuthenticator**(*username*, *password*)
Bases: *smtpproto.auth.SMTPAuthenticator*

Authenticates against the server with a username/password combination using the LOGIN method.

Parameters

- **username** (str) – user name to authenticate as
- **password** (str) – password to authenticate with

authenticate()
Performs authentication against the SMTP server.

This method must return an async generator. Any non-empty values the generator yields are sent to the server as authentication data. The response messages from any 334 responses are sent to the generator.

Return type AsyncGenerator[str, str]

property mechanism
The name of the authentication mechanism (e.g. PLAIN or GSSAPI).

Return type str

class smtpproto.auth.**OAuth2Authenticator**(*username*)
Bases: *smtpproto.auth.SMTPAuthenticator*

Authenticates against the server using OAUTH2.

In order to use this authenticator, you must subclass it and implement the *get_token()* method.

Parameters **username** (str) – the user name to authenticate as

authenticate()
Performs authentication against the SMTP server.

This method must return an async generator. Any non-empty values the generator yields are sent to the server as authentication data. The response messages from any 334 responses are sent to the generator.

Return type AsyncGenerator[str, str]

abstract async get_token()
Obtain a new access token.

Implementors should cache the token and its expiration time and only obtain a new one if the old one has expired or is about to.

Return type str

Returns the access token

property mechanism
The name of the authentication mechanism (e.g. PLAIN or GSSAPI).

Return type str

class smtpproto.auth.**PlainAuthenticator**(*username*, *password*, *authorization_id=''*)
Bases: *smtpproto.auth.SMTPAuthenticator*

Authenticates against the server with a username/password combination using the PLAIN method.

Parameters

- **username** (str) – user name to authenticate as

- **password** (str) – password to authenticate with
- **authorization_id** (str) – optional authorization ID

authenticate()
Performs authentication against the SMTP server.

This method must return an async generator. Any non-empty values the generator yields are sent to the server as authentication data. The response messages from any 334 responses are sent to the generator.

Return type AsyncGenerator[str, str]

property mechanism
The name of the authentication mechanism (e.g. PLAIN or GSSAPI).

Return type str

class smtpproto.auth.**SMTPAuthenticator**
Bases: object

Interface for providing credentials for authenticating against SMTP servers.

abstract authenticate()
Performs authentication against the SMTP server.

This method must return an async generator. Any non-empty values the generator yields are sent to the server as authentication data. The response messages from any 334 responses are sent to the generator.

Return type AsyncGenerator[str, str]

abstract property mechanism
The name of the authentication mechanism (e.g. PLAIN or GSSAPI).

Return type str

1.4.3 Concrete client implementation

class smtpproto.client.**AsyncSMTPClient**(*host*, *port=587*, *connect_timeout=30*, *timeout=60*, *domain=<factory>*, *ssl_context=None*, *authenticator=None*)
Bases: anyio.abc.AsyncResource

An asynchronous SMTP client.

This runs on asyncio or any other backend supported by AnyIO.

It is recommended that this client is used as an async context manager instead of manually calling *connect()* and *aclose()*, if possible.

Parameters

- **host** (str) – host name or IP address of the SMTP server
- **port** (int) – port on the SMTP server to connect to
- **connect_timeout** (float) – connection timeout (in seconds)
- **timeout** (float) – timeout for sending requests and reading responses (in seconds)
- **domain** (str) – domain name to send to the server as part of the greeting message
- **ssl_context** (Optional[SSLContext]) – SSL context to use for establishing TLS encrypted sessions

- **authenticator** (Optional[*SMTPAuthenticator*]) – authenticator to use for authenticating with the SMTP server

async aclose()
Close the connection, if connected.

Return type None

async connect()
Connect to the SMTP server.

Return type None

class smtpproto.client.**SyncSMTPClient**(**args*, *async_backend='asyncio'*, *async_backend_options=None*, ***kwargs*)

Bases: object

A synchronous (blocking) SMTP client.

It is recommended that this client is used as a context manager instead of manually calling *connect()* and *close()*, if possible.

Parameters

- **host** – host name or IP address of the SMTP server
- **port** – port on the SMTP server to connect to
- **connect_timeout** – connection timeout (in seconds)
- **timeout** – timeout for sending requests and reading responses (in seconds)
- **domain** – domain name to send to the server as part of the greeting message
- **ssl_context** – SSL context to use for establishing TLS encrypted sessions
- **authenticator** – authenticator to use for authenticating with the SMTP server
- **async_backend** (str) – name of the AnyIO-supported asynchronous backend
- **async_backend_options** (Optional[Dict[str, Any]]) – dictionary of keyword arguments passed to anyio.start_blocking_portal()

close()
Close the connection, if connected.

Return type None

connect()
Connect to the SMTP server.

Return type None

1.5 Version history

This library adheres to Semantic Versioning 2.0.

1.1.0

- Added missing authorization_id parameter to PlainAuthenticator (also fixes PLAIN authentication not working since this field was missing from the encoded output)
- Fixed sender/recipient addresses (in MAIL/RCPT commands) not being UTF-8 encoded in the presence of the SMTPUTF8 extension

1.0.0

- Initial release

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX

N

O

P

Q

R

S