TLS/SSL and PyMongo

PyMongo supports connecting to MongoDB over TLS/SSL. This guide covers the configuration options supported by PyMongo. See the server documentation to configure MongoDB.

Dependencies

For connections using TLS/SSL, PyMongo may require third party dependencies as determined by your version of Python. With PyMongo 3.3+, you can install PyMongo 3.3+ and any TLS/SSL-related dependencies using the following pip command:

```
$ python -m pip install pymongo[tls]
```

Earlier versions of PyMongo require you to manually install the dependencies listed below.

Python 2.x

The ipaddress module is required on all platforms.

When using CPython < 2.7.9 or PyPy < 2.5.1:

- On Windows, the wincertstore module is required.
- On all other platforms, the certifi module is required.

Warning: Industry best practices recommend, and some regulations require, the use of TLS 1.1 or newer. Though no application changes are required for PyMongo to make use of the newest protocols, some operating systems or versions may not provide an OpenSSL version new enough to support them.

Users of macOS older than 10.13 (High Sierra) will need to install Python from python.org, homebrew, macports, or another similar source.

Users of Linux or other non-macOS Unix can check their OpenSSL version like this:

```
$ openssl version
```

If the version number is less than 1.0.1 support for TLS 1.1 or newer is not available. Contact your operating system vendor for a solution or upgrade to a newer distribution.

You can check your Python interpreter by installing the requests module and executing the following command:

```
python -c "import requests; print(requests.get('https://www.howsmyssl.com/a/check',
```

You should see "TLS 1.X" where X is >= 1.

You can read more about TLS versions and their security implications here:

https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet#Rule_-_Only_Support_Strong_Protocols

Basic configuration

In many cases connecting to MongoDB over TLS/SSL requires nothing more than passing `ssl=True` as a keyword argument to `MongoClient`:

```
>>> client = pymongo.MongoClient('example.com', ssl=True)
```

Or passing `ssl=true` in the URI:

```
>>> client = pymongo.MongoClient('mongodb://example.com/?ssl=true')
```

This configures PyMongo to connect to the server using TLS, verify the server's certificate and verify that the host you are attempting to connect to is listed by that certificate.

Certificate verification policy

By default, PyMongo is configured to require a certificate from the server when TLS is enabled. This is configurable using the *ssl_cert_reqs* option. To disable this requirement pass `ssl.CERT_NONE` as a keyword parameter:

```
>>> import ssl
>>> client = pymongo.MongoClient('example.com',
...                              ssl=True,
...                              ssl_cert_reqs=ssl.CERT_NONE)
```

Or, in the URI:

```
>>> uri = 'mongodb://example.com/?ssl=true&ssl_cert_reqs=CERT_NONE'
>>> client = pymongo.MongoClient(uri)
```

Specifying a CA file

In some cases you may want to configure PyMongo to use a specific set of CA certificates. This is most often the case when using "self-signed" server certificates. The *ssl_ca_certs* option takes a path to a CA file. It can be passed as a keyword argument:

```
>>> client = pymongo.MongoClient('example.com',
...                              ssl=True,
...                              ssl_ca_certs='/path/to/ca.pem')
```

Or, in the URI:

```
>>> uri = 'mongodb://example.com/?ssl=true&ssl_ca_certs=/path/to/ca.pem'
>>> client = pymongo.MongoClient(uri)
```

Specifying a certificate revocation list

Python 2.7.9+ (pypy 2.5.1+) and 3.4+ provide support for certificate revocation lists. The *ssl_crlfile* option takes a path to a CRL file. It can be passed as a keyword argument:

```
>>> client = pymongo.MongoClient('example.com',
...                              ssl=True,
...                              ssl_crlfile='/path/to/crl.pem')
```

Or, in the URI:

```
>>> uri = 'mongodb://example.com/?ssl=true&ssl_crlfile=/path/to/crl.pem'
>>> client = pymongo.MongoClient(uri)
```

Client certificates

PyMongo can be configured to present a client certificate using the *ssl_certfile* option:

```
>>> client = pymongo.MongoClient('example.com',
...                              ssl=True,
...                              ssl_certfile='/path/to/client.pem')
```

If the private key for the client certificate is stored in a separate file use the *ssl_keyfile* option:

```
>>> client = pymongo.MongoClient('example.com',
...                              ssl=True,
...                              ssl_certfile='/path/to/client.pem',
...                              ssl_keyfile='/path/to/key.pem')
```

Python 2.7.9+ (pypy 2.5.1+) and 3.3+ support providing a password or passphrase to decrypt encrypted private keys. Use the *ssl_pem_passphrase* option:

```
>>> client = pymongo.MongoClient('example.com',
...                              ssl=True,
...                              ssl_certfile='/path/to/client.pem',
...                              ssl_keyfile='/path/to/key.pem',
...                              ssl_pem_passphrase=<passphrase>)
```

These options can also be passed as part of the MongoDB URI.

Troubleshooting TLS Errors

TLS errors often fall into two categories, certificate verification failure or protocol version mismatch. An error message similar to the following means that OpenSSL was not able to verify the server's certificate:

```
[SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed
```

This often occurs because OpenSSL does not have access to the system's root certificates or the certificates are out of date. Linux users should ensure that they have the latest root certificate updates installed from their Linux vendor. macOS users using Python 3.6.0 or newer downloaded from python.org may have to run a script included with python to install root certificates:

```
open "/Applications/Python <YOUR PYTHON VERSION>/Install Certificates.command"
```

Users of older PyPy portable versions may have to set an environment variable to tell OpenSSL where to find root certificates. This is easily done using the certifi module from pypi:

```
$ pypy -m pip install certifi
$ export SSL_CERT_FILE=$(pypy -c "import certifi; print(certifi.where())")
```

An error message similar to the following message means that the OpenSSL version used by Python does not support a new enough TLS protocol to connect to the server:

```
[SSL: TLSV1_ALERT_PROTOCOL_VERSION] tlsv1 alert protocol version
```

Industry best practices recommend, and some regulations require, that older TLS protocols be disabled in some MongoDB deployments. Some deployments may disable TLS 1.0, others may disable TLS 1.0 and TLS 1.1. See the warning earlier in this document for troubleshooting steps and solutions.