Secure Connections Overview

Conceptual overview of data-in-transit encryption in MariaDB, discussing supported TLS libraries (OpenSSL, wolfSSL), protocol versions (`tls_version`), and certificate verification.

Prior to MariaDB 11.4, by default, MariaDB transmits data between the server and clients without encrypting it. This is generally acceptable when the server and client run on the same host or in networks where security is guaranteed through other means. However, in cases where the server and client exist on separate networks or they are in a high-risk network, the lack of encryption does introduce security concerns as a malicious actor could potentially eavesdrop on the traffic as it is sent over the network between them.

To mitigate this concern, MariaDB allows you to encrypt data in transit between the server and clients using the Transport Layer Security (TLS) protocol. TLS was formerly known as Secure Socket Layer (SSL), but strictly speaking the SSL protocol is a predecessor to TLS and, that version of the protocol is now considered insecure. The documentation still uses the term SSL often and for compatibility reasons TLS-related server system and status variables still use the prefix `ssl_`, but internally, MariaDB only supports its secure successors.

Checking MariaDB Server for TLS Support

In order for MariaDB Server to use TLS, it needs to be compiled with TLS support. All MariaDB packages distributed by MariaDB Foundation and MariaDB Corporation are compiled with TLS support.

If you aren't sure whether your MariaDB Server binary was compiled with TLS support, then you can check the value of the have_ssl system variable. For example:

```
SHOW GLOBAL VARIABLES LIKE 'have_ssl';
+---------------+----------+
| Variable_name | Value    |
+---------------+----------+
| have_ssl      | DISABLED |
+---------------+----------+
```

The possible values are:

- If it is `DISABLED`, then the server was compiled with TLS support, but TLS is not enabled.
- If it is `YES`, then the server was compiled with TLS support, and TLS is enabled.
- If it is `NO`, then the server was not compiled with TLS support.

TLS Libraries

When MariaDB is compiled with TLS and cryptography support, it is usually either statically linked with MariaDB's bundled TLS and cryptography library, which might be wolfSSL ↗ or yaSSL ↗, or dynamically linked with the system's TLS and cryptography library, which might be OpenSSL ↗, GnuTLS ↗, or Schannel ↗.

See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms.

TLS Protocol Versions

There are 4 versions of the TLS protocol:

- `TLSv1.0`
- `TLSv1.1`
- `TLSv1.2`
- `TLSv1.3`

Enabling Specific TLS Protocol Versions

In some cases, it might make sense to only enable specific TLS protocol versions. For example, it would make sense if your organization has to comply with a specific security standard. It would also make sense if a vulnerability is found in a specific TLS protocol version, and you would like to ensure that your server does not use the vulnerable protocol version.

The PCI DSS v3.2 recommends using a minimum protocol version of TLSv1.2.

On the **server** side, users can enable specific TLS protocol versions by setting the tls_version system variable. This system variable accepts a comma-separated list of TLS protocol versions. A TLS protocol version will only be enabled if it is present in this list. All other TLS protocol versions will not be permitted. This system variable can be specified as a command-line argument to mariadbd or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
tls_version = TLSv1.2,TLSv1.3
```

You can check which TLS protocol versions are enabled on a server by executing SHOW GLOBAL VARIABLES. For example:

```
SHOW GLOBAL VARIABLES LIKE 'tls_version';
```

On the **client** side, users can enable specific TLS protocol versions by setting the `--tls-version` option. This option accepts a comma-separated list of TLS protocol versions. A TLS protocol version will only be enabled if it is present in this list. All other TLS protocol versions will not be permitted. For example, to specify this option in a relevant client option group in an option file, you could set the following:

```
[client-mariadb]
...
tls_version = TLSv1.2,TLSv1.3
```

Or if you wanted to specify it on the command-line with the mariadb client, then you could execute something like this:

```
$ mariadb -u myuser -p -h myserver.mydomain.com \
   --ssl \
   --tls-version="TLSv1.2,TLSv1.3"
```

TLS Protocol Version Support

The TLS protocol versions that are supported depend on the underlying TLS library used by the specific MariaDB binary.

TLS Library	Supported TLS Protocol Versions
openSSL	TLSv1, TLSv1.1, TLSv1.2, TLSv1.3
wolfSSL	TLSv1, TLSv1.1, TLSv1.2, TLSv1.3
yaSSL	TLSv1, TLSv1.1
Schannel	TLSv1, TLSv1.1, TLSv1.2
GnuTLS	TLSv1, TLSv1.1, TLSv1.2, TLSv1.3

See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used by the server and by clients on each platform.

TLS Protocol Version Support in OpenSSL

MariaDB binaries built with the OpenSSL ↗ library (OpenSSL 1.0.1 ↗ or later) support TLSv1.1 and TLSv1.2 since MariaDB 5.5.41, MariaDB 10.0.15, and MariaDB 10.1.4.

MariaDB binaries built with the OpenSSL ↗ library (OpenSSL 1.1.1 ↗ or later) support TLSv1.3 since MariaDB 10.2.16 and MariaDB 10.3.8.

If your MariaDB Server binary is built with OpenSSL, then you can set the ssl_cipher system variable to values like `SSLv3` or `TLSv1.2` to allow all SSLv3.0 or all TLSv1.2 ciphers. However, this does not necessarily limit the protocol version to TLSv1.2. See MDEV-14101 for more information about that.

Note that the `TLSv1.3` ciphers cannot be excluded when using OpenSSL, even by using the ssl_cipher system variable. See Using TLSv1.3 for details.

SSLv3.0 is known to be vulnerable to the POODLE attack, so it should not be used. SSLv2.0 and SSLv3.0 are disabled for MariaDB Server binaries linked with OpenSSL since MariaDB 5.5.41, MariaDB 10.0.15, and MariaDB 10.1.4. If you are using a MariaDB version older than that and you cannot upgrade, then please see the section titled "SSL 3.0 Fallback protection" in OpenSSL Security Advisory - 15 Oct 2014.

TLS Protocol Version Support in wolfSSL

MariaDB binaries built with the bundled wolfSSL library support TLSv1.0, TLSv1.1, TLSv1.2, and TLSv1.3.

TLS Protocol Version Support in yaSSL

MariaDB binaries built with the bundled yaSSL library support SSLv3.0, TLSv1.0, and TLSv1.1.

SSLv3.0 is known to be vulnerable to the POODLE attack, so it should not be used. SSLv2.0 and SSLv3.0 are disabled for MariaDB Server binaries linked with yaSSL since MariaDB 5.5.41, MariaDB 10.0.15, and MariaDB 10.1.4.

TLS Protocol Version Support in Schannel

MariaDB binaries built with the Schannel library support different versions of TLS on different versions of Windows. See the Protocols in TLS/SSL (Schannel SSP) documentation from Microsoft to determine which versions of TLS are supported on each version of Windows.

TLS Protocol Version Support in GnuTLS

MariaDB binaries built with the GnuTLS library support TLSv1.0, TLSv1.1, TLSv1.2, and TLSv1.3.

Enabling TLS

See Securing Connections for Client and Server for information on how to enable TLS on the client and server.

Certificate Verification

Certificate verification is how TLS authenticates its connections by verifying that it is talking to who it says it is. There are multiple components to this verification process:

- Was the certificate signed by a trusted Certificate Authority (CA)?
- Is the certificate expired?
- Is the certificate on my Certificate Revocation List (CRL)?
- Does the certificate belong to who I believe that I'm communicating with?

Certificate Authorities (CAs)

Certificate Authorities (CAs) are entities that you trust to sign TLS certificates. Your organization might have its own internal CA, or it might use trusted third-party CAs.

CAs are specified on the server and client by using the ssl_ca and ssl_capath options.

The ssl_ca option defines a path to a PEM file that should contain one or more X509 certificates for trusted Certificate Authorities (CAs). This option requires that you use the absolute path, not a relative path.

The ssl_capath option defines a path to a directory that contains one or more PEM files that should each contain one X509 certificate for a trusted Certificate Authority (CA). This option requires that you use the absolute path, not a relative path. The ssl_capath option is only supported if the server or client was built with OpenSSL ↗, wolfSSL ↗, or yaSSL ↗. If the client was built with GnuTLS ↗ or Schannel ↗, then the ssl_capath option is not supported.

See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms.

The directory specified by ssl_capath needs to be run through the openssl rehash ↗ command. For example, if the following is configured:

```
ssl_capath=/etc/my.cnf.d/certificates/ca/
```

Then you would have to execute the following:

```
openssl rehash /etc/my.cnf.d/certificates/ca/
```

Requiring a Specific Certificate Authority (CA)

The server can require a specific Certificate Authority (CA) for a client if the client's user account has been defined with `REQUIRE ISSUER`. See Securing Connections for Client and Server: Requiring TLS for more information.

Certificate Revocation Lists (CRLs)

Certificate Revocation Lists (CRLs) are lists of certificates that have been revoked by the Certificate Authority (CA) before they were due to expire.

CRLs are specified on the server and client by using the ssl_crl and ssl_crlpath options.

The ssl_crl option defines a path to a PEM file that should contain one or more X509 revoked certificates. This option requires that you use the absolute path, not a relative path. For servers, the ssl_crl option is only valid if the server was built with OpenSSL. If the server was built with wolfSSL ↗ or yaSSL ↗, then the ssl_crl option is not supported. For clients, the ssl_crl option is only valid if the client was built with OpenSSL ↗ or Schannel ↗. Likewise, if the client was built with GnuTLS ↗, wolfSSL ↗ or yaSSL ↗, then the ssl_crl option is not supported.

The ssl_crlpath option defines a path to a directory that contains one or more PEM files that should each contain one revoked X509 certificate. This option requires that you use the absolute path, not a relative path. The ssl_crlpath option is only supported if the server or client was built with OpenSSL ↗. If the server was built with wolfSSL ↗ or yaSSL ↗, then the ssl_crlpath option is not supported. Likewise, if the client was built with GnuTLS ↗, Schannel ↗, wolfSSL ↗, or yaSSL ↗, then the ssl_crlpath option is not supported.

See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms.

The directory specified by ssl_crlpath needs to be run through the openssl rehash ↗ command. For example, if the following is configured:

```
ssl_crlpath=/etc/my.cnf.d/certificates/crl/
```

Then you would have to execute the following:

```
openssl rehash /etc/my.cnf.d/certificates/crl/
```

Server Certificate Verification

Normally the TLS implementation performs numerous checks to verify whether the certificate is valid. It should be within its validity period, not revoked, signed by a trusted certificate authority, belong to the host that uses it.

Clients and utilities enable all these checks by default. Last two checks can be disabled with an option, such as `disable-ssl-verify-server-cert`. Before MariaDB 11.3, all server certificate verification checks were disabled by default, and could be enabled with the `ssl-verify-server-cert` option.

To verify whether the server's certificate belong to server's host, clients and utilities will check the **Common Name (CN)** attribute located in the **Subject** field of the certificate against the server's host name and IP address. If the **Common Name (CN)** matches either of those, then the certificate is verified.

Server Certificate Verification with Subject Alternative Names (SANs)

The **Subject Alternative Name (SAN)** field, which is an X.509v3 extension, can also be used for server certificate verification, if it is present in the server certificate. This field is also sometimes called **subjectAltName**. When using a client or utility that supports server certificate verification with **subjectAltName** fields, if the server certificate contains any **subjectAltName** fields, then those fields will also be checked against the server's host name and IP address.

Whether server certificate verification with **subjectAltName** fields is supported depends on the underlying TLS library used by the client or utility.

See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms.

SAN Support with OpenSSL, wolfSSL, and yaSSL

For clients and utilities built with OpenSSL (OpenSSL 1.0.2 or later), support for server certificate verification with **subjectAltName** fields that contain the server's **host name** was added in MariaDB 10.1.23 and MariaDB 10.2.6. See MDEV-10594 for more information.

For clients and utilities built with OpenSSL (OpenSSL 1.0.2 or later), support for server certificate verification with **subjectAltName** fields that contain the server's **IP address** was added in MariaDB 10.1.39, MariaDB 10.2.24, MariaDB 10.3.15, and MariaDB 10.4.5. See MDEV-18131 for more information.

This support also applies to other TLS libraries that use OpenSSL's API. In OpenSSL's API, server certificate verification with **subjectAltName** fields depends on the X509_check_host and X509_check_ip functions. These functions are supported in the following TLS libraries:

- OpenSSL 1.0.2 or later
- wolfSSL

And they are **not** supported in the following TLS libraries:

- yaSSL

MariaDB's RPM packages were built with OpenSSL 1.0.1 on RHEL 7 and CentOS 7, even after OpenSSL 1.0.2 became available on those distributions. As a side effect, the clients and utilities bundled in these packages did not support server certificate verification with the **subjectAltName** field, even if the packages were installed on a system that had OpenSSL 1.0.2 installed. Starting with MariaDB MariaDB 10.1.39, MariaDB 10.2.23, MariaDB 10.3.14, and MariaDB 10.4.4, MariaDB's RPM packages on RHEL 7 and CentOS 7 are built with OpenSSL 1.0.2. See MDEV-18277 for more information.

SAN Support with Schannel

For clients and utilities linked with Schannel, support for server certificate verification with **subjectAltName** fields was added in MariaDB Connector/C 3.0.2. See CONC-250 for more information.

SAN Support with GnuTLS

For clients and utilities linked with GnuTLS, support for server certificate verification with **subjectAltName** fields was added in MariaDB Connector/C 3.0.0. See CONC-250 for more information.

Client Certificate Verification

The server verifies a client certificate by checking the client's known `SUBJECT` against the **Subject** attribute in the client's certificate. This is only done for user accounts that have been defined with `REQUIRE SUBJECT`. See Securing Connections for Client and Server: Requiring TLS for more information.

See Also

- Mission Impossible: Zero-Configuration SSL ↗ (mariadb.org)
- Securing Connections for Client and Server
- Using TLSv1.3
- Certificate Creation with OpenSSL
- Replication with Secure Connections
- Securing Communications in Galera Cluster
- SSL/TLS System Variables
- Data-at-Rest Encryption
- Cyberciti tutorial: How to setup MariaDB SSL and secure connections from clients ↗

Previous
Replication with Secure Connections

Next
Securing Connections for Client and Server

Products

Enterprise Platform

Community Server

Download MariaDB

Pricing

Support

Customer Login

Technical Support

Remote DBA

Professional Services

Resources

MariaDB Blog

Webinars

Customer Stories

MariaDB Events

Documentation

Developer Hub

Company

About MariaDB

Newsroom

Leadership

MariaDB Careers

Legal

Privacy Policy