Agenda:

- Introduction
- Reasons to deploy TLS
- Rolling out TLS
Introduction.

Me:
- Daniël van Eeden
- Senior Database Engineer

Booking.com:
- Part of Booking Holdings Inc. (NASDAQ: BKNG)
- Booking.com on average books 1,550,000 room nights every 24 hours
- More than 15,500 employees
- 198 offices around the world
- 1000’s of MySQL and MariaDB servers
- 100’s of replication topologies
Note: $\text{SSL} \approx \text{TLS}$.

SSLv2 > SSLv3 > TLSv1.0 > TLSv1.1 > TLSv1.2 > TLSv1.3

MySQL 5.6 and older supported TLSv1.0 only
MySQL 5.7 supports TLSv1.0, TLSv1.1 and TLSv1.2 (TLSv1.2 only with OpenSSL)
Support for TLSv1.1 and TLSv1.2 was added to MariaDB in 5.5 and 10.0

So what MySQL supports really is TLS, but many of the older options refer to it as SSL. OpenSSL and YaSSL have SSL in the name, but do support TLS (SSL support might even be turned off).

So when talking about this people often mean TLS when they say SSL.
Why deploy TLS: account management.

MySQL protects the password when logging in with `mysql_native_password`.

However any statement send over a connection that is not encrypted is readable by anyone who has access to the network packets (tcpdump, wireshark, etc)

`SET PASSWORD ...`, `UPDATE mysql.user ...PASSWORD('secret')...`, etc might all leak your password. The "solution" is to calculate the password hash on the client and then send it over the unencrypted solution. But as `mysql_native_password` doesn't use a salt this isn't very secure either. The other solution is to run an agent on the machine and send the password over HTTPS to the agent and then have the agent run the statement over a UNIX socket connection.

All in all, enabling TLS allows remote management of accounts in a secure way.
Why deploy TLS: Compliance & Application security.

- TLS protects any data your application might send to or read from the database
- This might be part of your controls for
  - PCI-DSS
  - SOx
  - GDPR
- Note that at least PCI-DSS dictates that you disable older protocols
  - use the tls_version setting in MySQL
Why deploy TLS: client certificates.

- Client certificates allow authentication based on client certificates in addition or instead of passwords.
- One problem is certificate revocation (minimal CRL support and no OCSP support)
Why deploy TLS: caching_sha2.

- More secure because it uses SHA256 instead of double SHA1
- More secure because it uses a salt
- Uses TLS or an RSA keypair
- Managing RSA keypairs is painful and/or insecure
Why deploy TLS: PAM authentication.

- Using Percona PAM plugin: pam_compat
- Needs plaintext password (depends on what PAM modules are used, etc)
Rolling out TLS.

1. Request TLS certificates from our internal service
2. Enable TLS on MySQL
3. DONE 😊

Easy isn't it?
Rolling out TLS: First try.

- Enable TLS on a set of servers
  - Add ssl_ca, ssl_cert and ssl_key to my.cnf and restart
  - Result: TLS available, but not used
  - Set mysql_ssl=1 on the DSN to enable TLS
  - This was working fine, until...
  - We rebuild DBD::mysql against MySQL 5.7 instead of 5.6
  - MySQL 5.7 changed the defaults, it uses ssl-mode=PREFERRED by default.
    - MySQL 5.6 only used TLS when explicitly configured
  - During rollout:
    - On 10%: everything ok
    - On 100%: everything broken, as the master became overloaded
  - Adding mysql_ssl=0 did not disable TLS
Rolling out TLS: YaSSL and OpenSSL.

- We were using MySQL 5.7 Community Edition, which comes with YaSSL
- YaSSL is a legacy product from WolfSSL
  - YaSSL doesn't support TLSv1.2
  - YaSSL doesn't use optimized CPU instructions (AVX2, SSE4, etc)
- Note that MySQL 5.7 Enterprise Edition comes with OpenSSL
- Because of license compatibility Oracle can't really distribute a GPL build with OpenSSL.
- We now rebuild the 5.7 RPMs with OpenSSL
- This issue is gone in MySQL 8.0 as Oracle added a license exception. Now all builds come with OpenSSL.
- In addition YaSSL was replaced with WolfSSL (but not used as default)
Rolling out TLS: DBD::mysql.

- So why did mysql_ssl=0 not disable TLS?
- If mysql_ssl=1 was set SSL settings were provided to libmysqlclient
- If this wasn't the case then it relied on the library default to not use TLS
- On MySQL 5.7 the library default changed to use TLS if available (ssl-mode=PREFERRED), so not providing anything could result in a TLS connection
- The ssl-mode setting was added in 5.6 and extended in 5.7
- ssl-mode in 5.7 can be set to DISABLED, PREFERRED, REQUIRED, VERIFY_CA or VERIFY_IDENTITY. On 5.6 only REQUIRED was a valid option.
- Note that MariaDB doesn’t use ssl-mode, but slightly changed the meaning of --ssl and --ssl-verify-server-cert
Rolling out TLS: DBD::mysql.

- So I started to patch and test DBD::mysql
- mysql_ssl=0 now disables TLS
- mysql_ssl=1 now requires TLS
- mysql_ssl=1,mysql_ssl_optional=1 now uses TLS if available

Code depending on the version it as build against, the client library version and the server version. Both MySQL and MariaDB. A wide range of versions (4.1 to 8.0 and all MariaDB versions)

Along the way I found some security bugs where connections to MySQL 8.0 and MariaDB 10.x could be made over a non-secure connection even with the most strict settings.

I also became a co-maintainer for DBD::mysql
Rolling out TLS: Steps.

- First step is to enable TLS on slaves (or a percentage of slaves)
- Next step is to enable TLS on intermediate masters
- Now promote a intermediate master to a master
- ... and hope it works...
- If not: `gdb -p $(pidof mysqld) -ex "set ssl_acceptor_fd=0x0" -batch`
- In MySQL 8.0.16 and newer you can dynamically disable/enable TLS, no restart or GDB required.
- Applications are set to use TLS if available by default.
Rolling out TLS: Monitoring.

- Use performance_schema
  - `performance_schema.status_by_thread` has all the usual things like `Ssl_version`, `Ssl_cipher`, etc.
- This caused 5.7.21 and earlier to crash sporadically
- Besides monitoring the number of connections you might also want to monitor the connection rate.
- Monitor the expiry date of the certificates (not just the one on disk!)
Operational: Reloading certificates.

- MySQL 8.0 and MariaDB 10.4 support online reloading of certificates
- With MySQL 8.0 you can also disable TLS online.
- But you should restart MySQL every so often anyway to follow the Oracle Critical Patch Updates.
- This relies on a system that automates certificate management
Rolling out TLS: end user access.

- We have an internal service that allows people to run ad-hoc SQL queries
- This means MySQL Workbench, Sequel Pro, Excel, Tableau, etc.
- This uses Percona’s auth_pam_compat
- This means using the cleartext password client plugin
- This means we need to send the password to the server (instead of special nonce based handshake mysql_native_password) uses.
- So this requires TLS
- Additionally these applications don't know how to connect to a pool of servers
Rolling out TLS: end user access.

- Solution: mysqlredirect, a custom proxy written in Go
- This sets up a TLS connection between:
  - the client and the proxy
  - the proxy and the backend
- Anything else is transported as-is.
- This allows us to have a pool of servers as backend
- This allows us to have a TLS certificate that matches the hostname of the proxy
- This allows us to do strict TLS validation on the frontend and on the backend.
- This is not open source. Use ProxySQL, at the time ProxySQL didn’t have good TLS support yet.
Rolling out TLS: Certificate validation.

This brings us to Certificate Validation.

Things to validate:
- Not After date
- CA
- Hostname(s)
Rolling out TLS: Certificate validation.

But this wasn’t always working as expected:

- **Hostname**
  - Not done unless you set ssl-mode to VERIFY_IDENTITY
  - Wasn’t possible in Workbench before 6.3.9
  - Support for SubjectAlternativeName was added in 5.7.23 (based on my patch)
- **Not After date**
  - Fixed in 5.7.18 (and Workbench 8.0.12)
- **CA**
  - Note done unless you set ssl-mode to VERIFY_CA or VERIFY_IDENTITY
  - Before 5.7.18: If VERIFY_CA was set but no CA was specified it didn’t fail.
Enforcing TLS.

- Per account
  - REQUIRE SSL
  - REQUIRE X509 (SSL + valid certificate)
  - REQUIRE SUBJECT
  - REQUIRE ISSUER
  - REQUIRE CIPHER

- On the server: require_secure_transport
  - TLS or UNIX Domain Socket

- On the client:
  - requireTLS=true (Connector/J)
  - --ssl-mode=REQUIRED|VERIFY_CA|VERIFY.IDENTITY
  - mysql_ssl=1, etc (Perl)

Remember the auth_pam_compat with cleartext-plugin? We use require_secure_transport for that.
Rolling out TLS: percona-toolkit.

- So we have some servers with require_secure_transport
- We are in the process of rolling out TLS, not enabled everywhere just yet.
- We feed pt-online-schema-change a list of machines to monitor
- Now pt-osc connects, this fails on servers with require_secure_transport
- Setting mysql_ssl=1 causes it to fail on non-TLS servers
- Upgrading DBD::mysql and using mysql_ssl=1,mysql_ssl_optional=1 fixes the issue
Rolling out TLS: Orchestrator and go-mysql-driver.

- Orchestrator
  - We use GitHub Orchestrator
  - Orchestrator needs to connect to machines.
  - We taught it to re-connect over TLS when a host uses require-secure-transport
  - Orchestrator stores this per machine, to avoid re-connects.
  - Maintaining this cache wasn't without issues.

- Go
  - I added support for optional TLS: "tls=preferred" on the DSN
  - This should make it easier to support use TLS while it is not yet available on all servers.
Rolling out TLS: Java.

- We have a PEM CA certificate
- For Connector/Java we have to convert this to JKS before we can use it.
Rolling out TLS: RSA vs EC.

- I started to test connect times
- EC: ECDH-RSA-AES128-GCM-SHA256
- RSA: DHE-RSA-AES128-GCM-SHA256
- Connect time was about 50% faster with EC on MySQL 5.7 with OpenSSL

```
$ openssl ciphers -v 'DHE-RSA-AES128-GCM-SHA256:ECDH-RSA-AES128-GCM-SHA256'
DHE-RSA-AES128-GCM-SHA256  TLSv1.2  Kx=DH  Au=RSA  Enc=AESGCM(128)  Mac=AEAD
ECDH-RSA-AES128-GCM-SHA256  TLSv1.2  Kx=ECDH/RSA  Au=ECDH  Enc=AESGCM(128)  Mac=AEAD
```
Rolling out TLS: EC.

- For Java we use conscrypt as crypto library
- However conscrypt and MySQL with OpenSSL and EC don’t agree on ciphers
- A similar issue happens with golang and crypto/tls
- So EC is new and has some rough edges
- Running MySQL with a newer OpenSSL version might help
Rolling out TLS: Current status.

- Most servers run with TLS enabled
- More than 50% of connections actually use TLS
- Millions of concurrent TLS connections
Learnings

1. Use OpenSSL, don’t use YaSSL (basically: use MySQL 8.0 or a recent MariaDB)
2. Focus on enabling TLS on all MySQL Servers
3. Disable for specific connections if needed
4. Use TLS unless you have a strong reason not to (unless the other way around)
5. With TLS you are more future proof (caching_sha2)
Future

- TLSv1.3 0-RTT(-like)?
- Better support for EC ciphers
- Using TLS Tickets (Bug #76921 Resume SSL / TLS sessions (use TLS tickets))
  - 1000 connects: from 37s to 26s
- Increasing use of a Proxy
Questions?

Daniël van Eeden
daniel.vaneeden@booking.com
MariaDB specific features.

- Support for GnuTLS
- Support for SChannel (Windows only)
- Fingerprint verification
- Passphrase protected private keys
Wireshark

- Use a non-DH key
- Give wireshark your key
- It can now decrypt your traffic