Encrypt Replication Traffic

Background

MySQL supports using SSL to secure connections, including replication. (It can also be used for regular application connections, e.g. from PHP.)

Using SSL with MySQL can provide three benefits:

1. It can encrypt the data in transit, protecting us from snooping on the wire.
2. It can authenticate the server, so the client knows it is receiving authentic data.
3. It can authenticate the client, augmenting the password with public key cryptography.

Replication is excellent for getting a real-time snapshot of your data far away from the original database—the further the replica, the less likely it can be affected by the same outage. But for many organizations, traffic between datacenters (or between one cloud provider's Regions) crosses the public Internet. If there's anything in your database you wouldn't want on the cover of the New York Times, you should encrypt your replication traffic.

In this procedure, we'll configure SSL-protected replication between a master server in Boston and a business-continuity replica in London.

Steps

1. Check MySQL for SSL Support

   SSH to `boston.example.com` and check if MySQL is installed (or compiled) with SSL support.

```
boston ~ $ mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.5.30 MySQL Community Server (GPL)

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
```

```
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SHOW VARIABLES LIKE 'have_ssl';
+---------------+----------+
| Variable_name | Value    |
+---------------+----------+
| have_ssl      | DISABLED |
+---------------+----------+
1 row in set (0.00 sec)

mysql>
```

We expect a value of `DISABLED`. This means it is supported, but not configured, which we fix in Step 4.

A value of `YES` means it is supported and already configured.

A value of `NO` means it is not supported, and you'll need to reinstall or recompile MySQL, following these instructions.

After this step is completed:

1. On **Boston**, the MySQL variable `have_ssl` has value `DISABLED` or `YES`
2. On **London**, the MySQL variable `have_ssl` has value `DISABLED` or `YES`

2. Configure a Replication Account

Add a service account on **Boston** and give it replication privileges. The **London** server will use this account to authenticate to **Boston**.

```
mysql> GRANT REPLICATION SLAVE, REPLICATION CLIENT ON *.* TO 'replicator'@'%'
IDENTIFIED BY 'ToolGrubArmyBug';
Query OK, 0 rows affected (0.00 sec)

mysql> GRANT SELECT ON important.stuff TO 'replicator'@'%';
Query OK, 0 rows affected (0.00 sec)

mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

This grants SELECT rights to the `important.stuff` table, which aren't necessary for replication, but are necessary for the tests in Step 3 and Step 8.

After this step is completed:

1. `replicator` can log into MySQL on **Boston** from **London** with password `ToolGrubArmyBug`

3. Show Unencrypted Traffic

In this step, we'll capture traffic to show that without SSL, database traffic can be intercepted and read over the network. In a hurry? Skip to the next step.

With these settings, the replication account can connect from **London** to **Boston**, and MySQL traffic will not be encrypted.

SSH to `london.example.com`

Start a tcpdump session on **London**, capturing MySQL traffic to and from **Boston**

```
london ~ $ sudo tcpdump -ns 0 host boston.example.com and port 3306 -w /tmp/repl.pcap &
[1] 18796
london ~ $ tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
```

The output from tcpdump starting up obscured your prompt, press enter a few times, then connect to MySQL on **Boston** using the replicator account.

```
london ~ $
london ~ $
london ~ $ mysql -u replicator -p'ToolGrubArmyBug' -hboston.example.com
```

View the content of the important.stuff database. In the lab it's filled with random phrases, but in production it could be credit cards, private diary entries, etc.

Make a note of the details of some row.

```
mysql> select * from important.stuff;
+-----+---------------------+---------------------+
| id  | details             | happened            |
+-----+---------------------+---------------------+
|   1 | tack seam turn eat  | 2012-04-16 17:43:02 |
|   2 | size thin win spin  | 2012-04-17 13:30:47 |
|   3 | your lost king beef | 2012-04-18 15:25:15 |
|   4 | soar tell monk sold | 2012-04-20 01:47:14 |
|   5 | sign fall eye roar  | 2012-04-21 00:53:43 |
......some content not shown
| 296 | milk mule newt man  | 2013-04-15 05:30:20 |
+-----+---------------------+---------------------+
296 rows in set (0.01 sec)

mysql> exit
Bye
```

We used & to start tcpdump as a background job. Look it up the job number.

```
london ~ $ jobs
[1]+  Running                 sudo tcpdump host boston.example.com and port 3306 > /tmp/repl.pca
```

It's listed as job *1*, bring that job from the background to the foreground with fg

```
london ~ $ fg 1
sudo tcpdump host boston.example.com and port 3306 > /tmp/repl.pcap
```

Stop tcpdump by pressing control + c

```
^C36 packets captured
40 packets received by filter
0 packets dropped by kernel
london ~ $
```

Now search the tcpdump output for the detail text you noted above. If that text is in the dump file, it was transmitted over the network unencrypted.

```
london ~ $ grep --text "milk mule newt man" /tmp/repl.pcap
x crew2013-03-18 02:44:02+277flap hugs many ink2013-03-19 11:38:54,278slug bold spin
rail2013-03-21 02:53:51*279deer your mat pig2013-03-22 11:04:34,280cozy they crop
drip2013-03-23 18:05:28,281skin span poor debt2013-03-25 09:00:25,282jazz fair pill
aqua2013-03-26 21:21:06, 283foam boot they deck2013-03-27 18:11:31,!284wolf memo turf
fool2013-03-29 15:59:56,"285knew they thin brow2013-03-30 18:45:58,#286zest dune dent
deep2013-03-31 17:09:19,$287real rent cart rake2013-04-02 06:32:01+%288rice prop glow
hum2013-04-04 00:37:05*&289bun cart cast did2013-04-05 00:05:20,'290poor twin dome
edge2013-04-06 23:23:34,(291gale cash fake were2013-04-07 12:49:16+)292beg dose peel
fame2013-04-08 13:47:31,*293they chop unit heap2013-04-10 06:23:35,+294yoke rant torn
```

```
womb2013-04-12 05:26:06+,295play grow day tuck2013-04-13 22:17:46+-296milk mule newt
man2013-04-15 05:30:20.
london ~ $
```

Note that the tcpdump output file is in a binary format, so some characters will display as garbage or mojibake. This file format is designed to be re-used by the tcpdump interpreter (the `-r` flag) or with the Wireshark GUI.

4. Create SSL Keys for Boston

   On the **Boston** server, create a directory to store the Certificates

   ```
   mysql> exit
   Bye
   boston ~ $ sudo mkdir /etc/ssl/certs/mysql
   boston ~ $ sudo chown anonymous /etc/ssl/certs/mysql
   boston ~ $ cd /etc/ssl/certs/mysql
   ```

   Create an SSL Key Pair for Boston.

   `boston-private.pem` is the private key **Boston** will use to decrypt traffic. This must remain a secret to **Boston** alone.

   `boston-public.pem` is the public key **Boston** will give to clients (like **London**). Clients will use this key to authenticate **Boston**'s identity, and to encrypt traffic.

   ```
   boston /etc/ssl/certs/mysql $ openssl req -x509 -newkey rsa:1024 \
   -keyout boston-private.pem -out boston-public.pem \
   -subj '/CN=boston.example.com'  -nodes -days 3650
   Generating a 1024 bit RSA private key
   ............++++++
   ..............................++++++
   writing new private key to 'boston-private.pem'
   -----
   ```

   If two parties do not trust each other directly (e.g., your browser and your bank's web site), they both turn to a mutually trusted Certificate Authority. The CA validates that the server is who they claim to be by signing their certificate.

   In MySQL replication, the administrator controls both endpoints. Instead of trusting a CA, **London** will trust **Boston**'s public key explicitly.

   Create a new CA certificate file that contains a copy of **Boston's** certificate.

   ```
   boston /etc/ssl/certs/mysql $ cp boston-public.pem ca-cert.pem
   boston /etc/ssl/certs/mysql $
   ```

   After this step is completed:

   1. On **Boston**, a private key exists at `/etc/ssl/certs/mysql/boston-private.pem`
   2. On **Boston**, a public key exists at `/etc/ssl/certs/mysql/boston-public.pem`
   3. On **Boston**, a CA certificate exists at `/etc/ssl/certs/mysql/ca-cert.pem`

5. Configure Boston MySQL

   Configure Boston, both to be a replication master (see Establish Replication for details) and to support the new SSL certificates.

   `ssl-ca` - **Boston** will only accept certificates signed by (or in our case, contained in) this Certificate Authority (CA) certificate.

   `ssl-cert` - This is the public key the **Boston** server will use to represent itself to

London. **London** will use this to authenticate **Boston** and to encrypt traffic.

`ssl-key` - This is the private key **Boston** will use to decrypt information from **London**.

Edit my.cnf

```
boston ~ $ sudoedit /etc/my.cnf
```

Add these 5 lines

```
/etc/my.cnf

[mysqld]
log_bin = mysql-bin
server_id = 10
ssl-ca=/etc/ssl/certs/mysql/ca-cert.pem
ssl-cert=/etc/ssl/certs/mysql/boston-public.pem
ssl-key=/etc/ssl/certs/mysql/boston-private.pem
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
......some content not shown
```

The restart the MySQL process

```
boston ~ $ sudo service mysqld restart
Stopping mysqld:                                           [  OK  ]
Starting mysqld:                                           [  OK  ]
```

After this step is completed:

1. On **Boston**, `/etc/my.cnf` sets `server_id` to `10`
2. On **Boston**, `/etc/my.cnf` sets `log_bin` to `mysql-bin`
3. On **Boston**, `/etc/my.cnf` sets `ssl-ca` to `/etc/ssl/certs/mysql/boston-public.pem`
4. On **Boston**, `/etc/my.cnf` sets `ssl-cert` to `/etc/ssl/certs/mysql/boston-public.pem`
5. On **Boston**, `/etc/my.cnf` sets `ssl-key` to `/etc/ssl/certs/mysql/boston-private.pem`
6. On **Boston**, the MySQL variable `server_id` has value `10`
7. On **Boston**, the MySQL variable `log_bin` has value `ON`
8. On **Boston**, the MySQL variable `ssl-ca` has value `/etc/ssl/certs/mysql/boston-public.pem`
9. On **Boston**, the MySQL variable `ssl-cert` has value `/etc/ssl/certs/mysql/boston-public.pem`
10. On **Boston**, the MySQL variable `ssl-key` has value `/etc/ssl/certs/mysql/boston-private.pem`

6. Change Replicator user to Require SSL Encryption

Still on **Boston**, edit the replicator account to `REQUIRE SSL`

```
boston ~ $ mysql -u root
mysql> GRANT USAGE ON *.* TO 'replicator'@'%' REQUIRE SSL;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW GRANTS FOR "replicator";
+-----------------------------------------------------------------------------------------------+
| Grants for replicator@%
+-----------------------------------------------------------------------------------------------+
| GRANT REPLICATION SLAVE, REPLICATION CLIENT ON *.* TO 'replicator'@'%' IDENTIFIED BY PASSWORD
| GRANT SELECT ON `important`.`stuff` TO 'replicator'@'%'
+-----------------------------------------------------------------------------------------------+
2 rows in set (0.00 sec)

mysql>
```

Now, on the **London** server, try to connect to **Boston**. Connections that do not request SSL will fail:

```
london ~ $ mysql -u replicator -p'ToolGrubArmyBug' -hboston.example.com
ERROR 1045 (28000): Access denied for user 'replicator'@'domU-12-31-39-10-54-BD.compute-1.interr
```

After this step is completed:

1. On **Boston**, the `replicator` user's GRANT contains `REQUIRE SSL`
2. `replicator` can **not** log into MySQL on **Boston** from **London** with password `ToolGrubArmyBug` without SSL

7. Add CA Certificate to London

London needs a Certificate Authority (CA) to use SSL to connect to **Boston**.

Create a Place to store the Certificate:

```
london ~ $ sudo mkdir /etc/ssl/certs/mysql
london ~ $ sudo chown anonymous /etc/ssl/certs/mysql
london ~ $ cd /etc/ssl/certs/mysql
```

And copy the CA certificate from **Boston**:

```
london /etc/ssl/certs/mysql $ scp boston.example.com:/etc/ssl/certs/mysql/ca-cert.pem .
The authenticity of host 'boston.example.com (10.201.219.242)' can't be established.
RSA key fingerprint is db:42:29:45:dd:b3:ef:ff:1b:af:e8:10:b7:d5:29:ed.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'boston.example.com,10.201.219.242' (RSA) to the list of known hosts.
anonymous@boston.example.com's password: (input your password)
ca-cert.pem                                   100% 1718     1.7KB/s   00:00
london /etc/ssl/certs/mysql $
```

Now connect from **London** to **Boston**, using this CA certificate to authenticate **Boston**'s identity:

```
london ~ $ mysql -u replicator -p'ToolGrubArmyBug' -hboston.example.com \
    --ssl-ca /etc/ssl/certs/mysql/ca-cert.pem --ssl-verify-server
```

Once you connect, check the `Ssl_cipher` status variable. The connection is encrypted, `Ssl_cipher` will show what cypher is in use. (Default would be blank.)

```
mysql> SHOW STATUS LIKE 'Ssl_cipher';
+---------------+--------------------+
| Variable_name | Value              |
+---------------+--------------------+
| Ssl_cipher    | DHE-RSA-AES256-SHA |
+---------------+--------------------+
```

After this step is completed:

1. On **London**, a CA certificate exists at `/etc/ssl/certs/mysql/ca-cert.pem`
2. `replicator` can log into MySQL *with SSL* from **London** to **Boston** with password `ToolGrubArmyBug`
3. The connection from **London** to **Boston** *is encrypted*, state variable `Ssl_cipher` is not blank.

8. Show Encrypted Traffic

In this step, we'll capture traffic again to show that database traffic is encrypted. In a hurry? Skip to the next step.

First, set up another `tcpdump`

```
mysql> exit
Bye
london ~ $ sudo tcpdump -ns 0 host boston.example.com and port 3306 -w /tmp/repl.pcap &
[1] 18796
london ~ $ tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
```

Press `enter` a few times, then connect to MySQL on **Boston** using the replicator account and the new CA certificate.

```
london ~ $
london ~ $
london ~ $ mysql -u replicator -p'ToolGrubArmyBug' -hboston.example.com \
    --ssl-ca /etc/ssl/certs/mysql/ca-cert.pem --ssl-verify-server
```

Again, view the content in the `important.stuff` database, and again, make a note of the details of some row.

```
mysql> select * from important.stuff;
+-----+---------------------+---------------------+
| id  | details             | happened            |
+-----+---------------------+---------------------+
|   1 | tack seam turn eat  | 2012-04-16 17:43:02 |
|   2 | size thin win spin  | 2012-04-17 13:30:47 |
|   3 | your lost king beef | 2012-04-18 15:25:15 |
|   4 | soar tell monk sold | 2012-04-20 01:47:14 |
|   5 | sign fall eye roar  | 2012-04-21 00:53:43 |
......some content not shown
| 296 | milk mule newt man  | 2013-04-15 05:30:20 |
+-----+---------------------+---------------------+
296 rows in set (0.01 sec)

mysql> exit
Bye
```

Look up the job number `tcpdump` started with.

```
london ~ $ jobs
[1]+  Running                 sudo tcpdump host boston.example.com and port 3306 > /tmp/repl.pca
```

Bring that job to the foreground, and press `control` + `c` to end it.

```
london ~ $ fg 1
sudo tcpdump host boston.example.com and port 3306 > /tmp/repl.pcap
^C36 packets captured
40 packets received by filter
0 packets dropped by kernel
london ~ $
```

This time when you search the tcpdump output for the detail text you noted above, you won't find it. The connection is now encrypted.

```
london ~ $ grep --text "milk mule newt man" /tmp/repl.pcap
london ~ $
```

9. Change Replicator user to Verify London's Public Key

   On **Boston**, edit the replicator account's SSL REQUIRE statement to look for a specific certificate Subject.

```
mysql> GRANT USAGE ON *.* TO 'replicator'@'%'
REQUIRE SUBJECT '/CN=london.example.com';
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW GRANTS FOR "replicator";
+-----------------------------------------------------------------------------------------------
| Grants for replicator@%
```

```
+-----------------------------------------------------------------------------------------------
| GRANT REPLICATION SLAVE, REPLICATION CLIENT ON *.* TO 'replicator'@'%' IDENTIFIED BY PASSWORD
| GRANT SELECT ON `important`.`stuff` TO 'replicator'@'%'
+-----------------------------------------------------------------------------------------------
2 rows in set (0.00 sec)

mysql>
```

Now, connections from **London** to **Boston** will fail, because **London** does not have a certificate with that Subject:

```
london ~ $ mysql -u replicator -p'ToolGrubArmyBug' -hboston.example.com \
    --ssl-ca /etc/ssl/certs/mysql/ca-cert.pem --ssl-verify-server
ERROR 1045 (28000): Access denied for user 'replicator'@'domU-12-31-39-06-26-67.compute-1.interr
```

Note that the connection fails before **Boston** evaluates whether **London** provided the correct password.

After this step is completed:

1. On **Boston**, the `replicator` user's GRANT contains `REQUIRE SUBJECT '/CN=london.example.com'`
2. `replicator` can **not** log into MySQL on **Boston** from **London** with password `ToolGrubArmyBug` without a trusted key.

10. Create SSL Keys for London

Create a Key and Self-Signed Certificate for London. Note that the subject (`-subj`) is what **Boston** is already expecting.

`london-private.pem` is the private key **London** will use to decrypt traffic, and must remain a secret to **London** alone.

`london-public.pem` is the public key **Boston** will use to authenticate **London**'s identity, and to encrypt traffic.

```
london /etc/ssl/certs/mysql $ openssl req -x509 -newkey rsa:1024 \
-keyout london-private.pem -out london-public.pem \
-subj '/CN=london.example.com'  -nodes -days 3650
Generating a 1024 bit RSA private key
.............++++++
.............................++++++
writing new private key to 'london-private.pem'
-----
```

Append the new London certificate into the CA certificate.

```
london /etc/ssl/certs/mysql $ cat london-public.pem >> ca-cert.pem
```

And copy the amended CA certificate back to **Boston**:

```
london /etc/ssl/certs/mysql $ scp ca-cert.pem boston.example.com:/etc/ssl/certs/mysql/
anonymous@boston.example.com's password: (input your password)
ca-cert.pem                                   100% 1718     1.7KB/s   00:00
london /etc/ssl/certs/mysql $
```

Boston needs a MySQL restart to load the changes to `ca-cert`

```
boston ~ $ sudo service mysqld restart
Stopping mysqld:                                           [  OK  ]
Starting mysqld:                                           [  OK  ]
```

After this step is completed:

1. On **London**, a private key exists at `/etc/ssl/certs/mysql/boston-private.pem`
2. On **London**, a public key exists at `/etc/ssl/certs/mysql/boston-public.pem`
3. On **London**, a CA certificate exists at `/etc/ssl/certs/mysql/ca-cert.pem`

11. Secure the SSL Keys

    The key files have all been owned by `anonymous` to this point to make creating and copying them easier.

    That's not a good idea long term, so tighten up the file permissions.

    The certificates will be owned by the `mysql` user, (the same headless account the MySQL process runs as). Because in normal operation the certificates don't change, they will be marked read-only—if the certificates need to be replaced or altered later, use `sudo` to intentionally escalate privileges, with appropriate logging and access controls.

    Run this procedure on both servers:

```
london ~ $ sudo chown -R mysql:mysql /etc/ssl/certs/mysql
london ~ $ sudo chmod a=r /etc/ssl/certs/mysql/*
london ~ $ ls -l /etc/ssl/certs/mysql
total 12
-r--r--r-- 1 mysql mysql  916 Apr 17 19:19 boston-private.pem
-r--r--r-- 1 mysql mysql  790 Apr 17 19:19 boston-public.pem
-r--r--r-- 1 mysql mysql 1580 Apr 17 19:30 ca-cert.pem
london ~ $ sudo chmod a=rx /etc/ssl/certs/mysql
london ~ $ ls -ld /etc/ssl/certs/mysql
dr-xr-xr-x 2 mysql mysql 4096 Apr 17 19:20 /etc/ssl/certs/mysql
```

    In production, you may choose to make the files only readable by the `mysql` user (`u=r,go=`). In the lab, the student account still needs to read the certificates to use the interactive MySQL client.

    After this step is completed:

    1. On **Boston**, `/etc/ssl/certs/mysql/boston-private.pem` is owned by `mysql` with permissions `-r--r--r--`
    2. On **Boston**, `/etc/ssl/certs/mysql/boston-public.pem` is owned by `mysql` with permissions `-r--r--r--`
    3. On **Boston**, `/etc/ssl/certs/mysql/ca-cert.pem` is owned by `mysql` with permissions `-r--r--r--`
    4. On **London**, `/etc/ssl/certs/mysql/london-private.pem` is owned by `mysql` with permissions `-r--r--r--`
    5. On **London**, `/etc/ssl/certs/mysql/london-public.pem` is owned by `mysql` with permissions `-r--r--r--`
    6. On **London**, `/etc/ssl/certs/mysql/ca-cert.pem` is owned by `mysql` with permissions `-r--r--r--`

12. Connect from London to Boston with the Required Key

    Connect with MySQL client from **London** to **Boston** using new keys.

    A client can determine whether the current connection with the server uses SSL by checking the value of the Ssl_cipher status variable. The value of Ssl_cipher is nonempty if SSL is used, and empty otherwise. For example:

```
london ~ $ mysql -u replicator -p'ToolGrubArmyBug' -hboston.example.com \
    --ssl-ca /etc/ssl/certs/mysql/ca-cert.pem \
    --ssl-cert /etc/ssl/certs/mysql/london-public.pem \
    --ssl-key /etc/ssl/certs/mysql/london-private.pem
```

```
mysql> SHOW STATUS LIKE 'Ssl_cipher';
+---------------+--------------------+
| Variable_name | Value              |
+---------------+--------------------+
| Ssl_cipher    | DHE-RSA-AES256-SHA |
+---------------+--------------------+
```

After this step is completed:

1. `replicator` can log into MySQL on **Boston** from **London** with password `ToolGrubArmyBug` with a trusted key.

13. Back up Boston

    On **Boston**, make a complete backup, including replication master settings, and copy it to **London**.

```
boston ~ $ mysqldump -u root --single-transaction --all-databases --master-data=1 > /tmp/master_
-- Warning: Skipping the data of table mysql.event. Specify the --events option explicitly.
boston ~ $ scp /tmp/master_backup.sql london.example.com:/tmp/
The authenticity of host 'london.example.com (10.242.58.189)' can't be established.
RSA key fingerprint is 0f:47:42:f4:71:51:4c:a3:70:94:db:83:03:4c:d2:48.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'london.example.com,10.242.58.189' (RSA) to the list of known hosts.
anonymous@london.example.com's password: (input your password)
master_backup.sql                             100%  501KB 501.3KB/s   00:00
boston ~ $
```

After this step is completed:

1. The backup file exists on **Boston** at `/tmp/master_backup.sql`
2. The backup file exists on **London** at `/tmp/master_backup.sql`

14. Configure London to be a Slave

    Edit my.cnf

```
mysql> exit
london ~ $ sudoedit /etc/my.cnf
```

    Add these lines under the `[mysql]` heading. (See Establish Replication for descriptions and the process to verify they've taken effect.)

```
/etc/my.cnf

[mysqld]
log_bin = mysql-bin
server_id = 20
log_slave_updates = 1
relay_log = mysql-relay-bin
read_only = 1
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
......some content not shown
```

    Then restart the MySQL process

```
london ~ $ sudo service mysqld restart
Stopping mysqld:                                           [  OK  ]
Starting mysqld:                                           [  OK  ]
```

After this step is completed:

1. On **London**, `/etc/my.cnf` sets `server_id` to `20`

2. On **London**, `/etc/my.cnf` sets `log_bin` to `mysql-bin`
3. On **London**, `/etc/my.cnf` sets `log_slave_updates` to `1`
4. On **London**, `/etc/my.cnf` sets `relay_log` to `mysql-relay-bin`
5. On **London**, `/etc/my.cnf` sets `read_only` to `1`
6. On **London**, the MySQL variable `server_id` has value `20`
7. On **London**, the MySQL variable `log_bin` has value `ON`
8. On **London**, the MySQL variable `log_slave_updates` has value `ON`
9. On **London**, the MySQL variable `relay_log` has value `ON`
10. On **London**, the MySQL variable `read_only` has value `ON`

15. Start London Replicating from Boston

    Import the backup from **Boston**.

    You will get dozens of lines of status messages. Scan them briefly to make sure there are no errors.

```
london ~ $ mysql -u root
mysql> source /tmp/master_backup.sql
Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
......some content not shown
```

    Configure **London** to replicate from **Boston**.

    Most of these settings are the replication versions of the command line arguments you used in Step 12

```
mysql> CHANGE MASTER TO
Master_Host='boston.example.com',
Master_User='replicator',
Master_Password='ToolGrubArmyBug',
Master_SSL=1,
Master_SSL_CA = '/etc/ssl/certs/mysql/ca-cert.pem',
Master_SSL_CERT = '/etc/ssl/certs/mysql/london-public.pem',
Master_SSL_KEY = '/etc/ssl/certs/mysql/london-private.pem',
Master_SSL_Verify_Server_Cert = 1;
Query OK, 0 rows affected (0.04 sec)

mysql> slave start;
Query OK, 0 rows affected (0.00 sec)
```

    Confirm replication is running.

```
mysql> show slave status\G
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: boston.example.com
                  Master_User: replicator
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: mysql-bin.000002
          Read_Master_Log_Pos: 107
               Relay_Log_File: mysql-relay-bin.000003
                Relay_Log_Pos: 253
        Relay_Master_Log_File: mysql-bin.000002
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
```

```
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Master_Log_Pos: 107
              Relay_Log_Space: 844
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Master_SSL_Allowed: Yes
           Master_SSL_CA_File: /etc/ssl/certs/mysql/ca-cert.pem
           Master_SSL_CA_Path:
              Master_SSL_Cert: /etc/ssl/certs/mysql/london-public.pem
            Master_SSL_Cipher:
               Master_SSL_Key: /etc/ssl/certs/mysql/london-private.pem
        Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 0
                Last_IO_Error:
               Last_SQL_Errno: 0
               Last_SQL_Error:
  Replicate_Ignore_Server_Ids:
             Master_Server_Id: 10
1 row in set (0.00 sec)

mysql>
```

After this step is completed:

1. On **London**, MySQL has the database `important`
2. On **London**, MySQL has the table `important.stuff`
3. On **London**, the replication status for `Master_Host` is `boston.example.com`
4. On **London**, the replication status for `Master_User` is `replicator`
5. On **London**, the replication status for `Slave_IO_Running` is `Yes`
6. On **London**, the replication status for `Slave_SQL_Running` is `Yes`
7. On **London**, the replication status for `Master_SSL_Allowed` is `Yes`
8. On **London**, the replication status for `Master_SSL_CA_File` is `/etc/ssl/certs/mysql/ca-cert.pem`
9. On **London**, the replication status for `Master_SSL_Cert` is `/etc/ssl/certs/mysql/london-public.pem`
10. On **London**, the replication status for `Master_SSL_Key` is `/etc/ssl/certs/mysql/london-private.pem`

16. Verify Replication is Running and Encrypted

    Set up a new `tcpdump` session on **London**. This time we won't run it in the background:

    ```
    london ~ $ sudo tcpdump -s 0 -A -vv host boston.example.com and port 3306
    tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
    ```

    Now, insert some new data on the **Boston** server.

    ```
    boston ~ $ mysql -u root

    mysql> INSERT INTO important.stuff SET details='Replication is running encrypted';
    Query OK, 1 row affected (0.04 sec)

    mysql>
    ```

    The replication packets will appear on **London**'s tcpdump output, almost immediately.

    It's not much to look at, because the data is encrypted. The same `tcpdump` on an

unencrypted replication, would show the replicating statement in plain text.

```
21:01:56.285603 IP (tos 0x0, ttl 61, id 36926, offset 0, flags [DF], proto TCP (6), length
398)
    ip-10-110-179-3.ec2.internal.mysql > domU-12-31-39-06-26-67.compute-1.internal.49471:
Flags [P.], cksum 0x2e1d (correct), seq 3608204924:3608205270, ack 3612896711, win 142,
options [nop,nop,TS val 1997595 ecr 1948794], length 346
E....>@.=..Y
n..
.)....?...|.Xm............
..{....z.... 3......0.6.d.h..F.?...S.b|.+.y......0...9|.o;..2..|+.ty..5...D.>
<..H&.fxc...=y..S%.cU...C).........,..0.....M._.p        Q/x.N.u..`..>...[.f...f]6..
[.ef..`.e......8..?d.1.O4.)..s.Q.#....84..ip.l)..o,i.....................=....%.M..o.f...HAni....
2j..o.}..l3@Q"....N.5*..?Z...?c_.....i......L)..I&*...7.'3.
{..RL#sV...I..8a........CWo.|...!b..dIr
21:01:56.285659 IP (tos 0x8, ttl 64, id 54699, offset 0, flags [DF], proto TCP (6), length 52)
    domU-12-31-39-06-26-67.compute-1.internal.49471 > ip-10-110-179-3.ec2.internal.mysql:
Flags [.], cksum 0xf1f8 (incorrect -> 0x72d0), seq 1, ack 346, win 229, options [nop,nop,TS
val 1954733 ecr 1997595], length 0
E..4..@.@.s>
.).
n...?...Xm..............
......{.
```

Stop `tcpdump` by pressing `control` + `c`

```
^C2 packets captured
6 packets received by filter
0 packets dropped by kernel
london ~ $
```

Now log back into MySQL on **London** and verify the row arrived.

```
london ~ $ mysql -u root
mysql> SELECT * FROM important.stuff ORDER BY id DESC LIMIT 1;
+-----+---------------------------------+---------------------+
| id  | details                         | happened            |
+-----+---------------------------------+---------------------+
| 306 | Replication is running encrypted | 2013-04-17 20:17:17 |
+-----+---------------------------------+---------------------+
1 row in set (0.00 sec)

mysql>
```

After this step is completed:

1. INSERT a new record on **Boston** in the table `important.stuff`
2. Within 1 second, that record is replicated to **London**

"Don't take the author's word for anything; prove it to yourself. Do the exercises and invent your own." —James Hague

We're happy to provide this how-to for anyone who needs it. If you're in the thick of a problem, we'd like to help where we can and stay out of your way.