State of MySQL Security: 2022

Alexander Rubin
Senior Database Engineer
Amazon RDS
About me

• Working with MySQL for ~15 years
• Started at MySQL AB 2006
  • Sun Microsystems, Oracle (MySQL Consulting)
  • Percona since 2014
• Joined the Amazon Relational Database Service (RDS) engineering team in 2020
• Currently leading database security team
  • I’m hiring Pentesters/Security Engineers!
Amazon RDS
Set up, operate, and scale a relational database in the cloud with just a few clicks

Easy to administer
Secure and compliant
Available and durable
Performant and scalable

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.
How many of you use a specific MySQL version?
MySQL History and Security Features over the years

- **v. 3.23**
  - AuthN (users/pass)
  - Host base part of username
  - AuthZ (grants)

- **v. 4.0**
  - SSL option for connections

- **v. 4.1**
  - Better password hash

- **v. 5.0**
  - Audit plugin (enterprise)
  - SSL for replication
MySQL History and Security Features over the years

- V 5.1: sha256 password
- V 5.5: default ssl encryption
- V 5.6: encryption at rest
- V 5.7: caching sha256 password
- V 8.0: Secure defaults (Enterprise only features)
MySQL 8.0
Security feature outline

New security features:
• Encryption
  • In flight (SSL/TLS, client->server, replication, etc)
  • At rest (tablespace encryption, key management
  • Various cryptographic functions
• AuthN
  • New auth protocol:
  • New auth methods (Kerberos, etc)
  • Pluggable authentication
• AuthZ
  • Roles

Most important: “Secure defaults”
Over 3.6 million MySQL servers are publicly exposed on the Internet and responding to queries, making them an attractive target to hackers and extortionists.

Our agenda

Encryption
- In flight (client->server)
- At rest (on disk)

Authentication
- User/password

Authorization
- Grants/roles

Accounting
- Logs: audit log
Our agenda

- **Encryption**
  - In flight (client->server)
  - At rest (on disk)

- **Authentication**
  - User/password

- **Authorization**
  - Grants/roles

- **Accounting**
  - Logs: audit log

We are here
Data in Flight Encryption - why do we need it?

mysql -h 172.31.1.242 -P 5726 -umsandbox -pmsandbox
mysql>
mysql>
mysql>
mysql> \s
------------
mysql  Ver 14.14 Distrib 5.7.38, for Linux (x86_64) using EditLine wrapper

Connection id:  5
Current database:  creditcards
Current user:  msandbox@ip-172-31-1-242
SSL:  Not in use
SSL: Not in use
Current pager: stdout
Using outfile: '
Using delimiter: ;
Server version: 8.0.23 MySQL Community Server - GPL
Protocol version: 10
Connection: 172.31.1.242 via TCP/IP
Server charset: utf8mb4
Db charset: utf8mb4
Client charset: utf8
Conn. charset: utf8
TCP port: 8023
Uptime: 1 day 12 hours 50 min 36 sec

Threads: 2 Questions: 53 Slow queries: 0 Opens: 201 Flush tables: 3 Open tables: 120 Queries per second avg: 0.000

mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql> select * from cc;
Data in Flight Encryption - why do we need it?

mysql> SSL: Not in use
mysql> select * from cc;
+---------------------+
| cc_num              |
+---------------------+
| 1234-4564-0984-9874 |
+---------------------+
1 row in set (0.00 sec)

# tcpdump -i any -s 0 -l -w - port 8023 | strings
tcpdump: listening on any ... select * from cc
creditcards
cc_num
cc_num
1234-4564-0984-9874

Bad actor intercepted traffic with tcpdump!
SSL/TLS: Default in MySQL 5.7+

- If SSL is enabled (default) on the server client will use it
- No need to generate keys and send it to the client
  - Server key - will be generated when MySQL starts
  - Client key will be generated on demand
SSL/TLS: Default in MySQL 5.7+

$ mysql -h db
Welcome to the MySQL monitor. Commands end with ; or \g.
...
Server version: 5.7.25-28-57 (GPL)

mysql> \s
------------------
Connection id:          799621
...
SSL: Cipher in use is ECDHE-RSA-AES128-GCM-SHA256
Data in Flight Encryption - client connection

MySQL workbench:

Successfully made the MySQL connection

Information related to this connection:

Host: 127.0.0.1
Port: 3310
User: dev
SSL: enabled with ECDHE-RSA-AES128-GCM-SHA256

A successful MySQL connection was made with the parameters defined for this connection.

OK
SSL: Cipher in use is ECDHE-RSA-AES128-GCM-SHA256
Current pager: stdout
Using outfile: 
Using delimiter: ;
Server version: 8.0.23 MySQL Community Server - GPL
Protocol version: 10
Connection: 172.31.1.242 via TCP/IP
Server character set: utf8mb4
Db character set: utf8mb4
Client character set: utf8
Conn. character set: utf8
TCP port: 8023
Uptime: 1 day 12 hours 56 min 14 sec
Threads: 2 Questions: 62 Slow queries: 0 Opens: 201 Flush tables: 3 Open tables: 120 Queries per second avg: 0.000

mysql> 
mysql> 
mysql> 
mysql> 
mysql> 
mysql> 
mysql> 
mysql> 
mysql> 
mysql> 
mysql> select * from cc;
Data in Flight Encryption - why do we need it?

Cipher in use is ECDHE-RSA-AES128-GCM-SHA256

mysql> select * from cc;
+-----+---------------------+
| cc_num | 1234-4564-0984-9874 |
+-----+---------------------+
1 row in set (0.00 sec)

# tcpdump -i any -s 0 -l -w - port 8023|strings
tcpdump: listening on any ...

Protected!
Create user and force ssl/tls

CREATE USER 'user'@'<host>' IDENTIFIED BY '<pass here>' REQUIRE SSL;

$ mysql> alter user dev@'10.0.0.1' require ssl;
Query OK, 0 rows affected (0.00 sec)

$ mysql -u dev -h 10.0.0.1 -e '\s' | grep SSL
SSL: Cipher in use is TLS_AES_256_GCM_SHA384

$ mysql -u dev -h 10.0.0.1 --skip-ssl
ERROR 1045 (28000): Access denied for user 'dev'@'10.0.0.1' (using password: YES)
Data in Flight Encryption - server to server

Protecting communications:
• source -> replica
• between nodes in a cluster
• etc
Our agenda

Encryption
- In flight (client->server)
- At rest (on disk)

Authentication
- User/password

Authorization
- Grants/roles

Accounting
- Logs: audit log
Data at Rest Encryption – Why do we need it?

mysql> create table a(s varchar(255)) engine=InnoDB;
Query OK, 0 rows affected (0.01 sec)

mysql> insert into a values ('AlexanderRubin');
Query OK, 1 row affected (0.00 sec)

mysql> insert into a values ('qqqqqqq');
Query OK, 1 row affected (0.00 sec)

mysql> update a set s = 'AlexanderRubin';
Query OK, 1 row affected (0.00 sec)
Rows matched: 2  Changed: 1  Warnings: 0
Data at Rest Encryption – Why do we need it?

```
/data/mysql# grep -r 'AlexanderRubin' *
Binary file ib_logfile0 matches
Binary file log-bin.000004 matches
Binary file test/a.ibd matches
Binary file xb_doublewrite matches
```

PII data stored on disk in clear text
Data at Rest Encryption

Transparent Database Encryption (TDE): MySQL implementation

1. Create master key and store it
2. Use master key to encrypt table (tablespace) key
3. Use tablespace key to encrypt table data

More information:
Data at Rest Encryption

Transparent Database Encryption (TDE): encrypting db files

1. **InnoDB files**: tablespaces, redo logs, undo logs:
   - Available since MySQL 5.7

2. **Binary logs, relay logs**: for MySQL replication:
   - Available in MySQL 8.0 and Percona Server 5.7 & 8.0

3. **Tmp files**: Available in Percona Server 5.7 & 8.0

https://www.percona.com/doc/percona-server/5.7/management/data_at_rest_encryption.html


Data at Rest Encryption: add encryption options

[mysqld]
early-plugin-load=keyring_file.so
keyring_file_data=/mount/mysql/mysql-keyring/keyring
innodb_sys_tablespace_encrypt=1
innodb_parallel_dblwr_encrypt=1
innodb_temp_tablespace_encrypt=1
innodb_encrypt_tables=FORCE
innodb_encrypt_online_alter_logs=1
innodb_undo_log_encrypt=1
innodb_redo_log_encrypt=1
innodb_scrub_log=1
master_verify_checksum=1
binlog_checksum=1
encrypt_binlog=1
encrypt_tmp_files=1

Encrypt everything!
Data at Rest Encryption

mysql> create table a(s varchar(255)) engine=InnoDB /* encrypted='y' */;
Query OK, 0 rows affected (0.01 sec)

mysql> insert into a values ('AlexanderRubin');
Query OK, 1 row affected (0.00 sec)

mysql> insert into a values ('qqqqqqq');
Query OK, 1 row affected (0.00 sec)

mysql> update a set s = 'AlexanderRubin';
Query OK, 1 row affected (0.00 sec)
Rows matched: 2  Changed: 1  Warnings: 0
Data at Rest Encryption: add encryption options

/data/mysql# grep -r 'AlexanderRubin' *
/data/mysql#

Nothing found!
(data encrypted)
Our agenda

Encryption
• In flight (client->server)
• At rest (on disk)

Authentication
• User/password

Authorization
• Grants/roles

Accounting
• Logs: audit log

We are here
3 A’s of Security

Authentication
Authorization
Accounting

security framework that controls access to computer resources, enforces policies, and audits usage.
# 3 A’s in MySQL Security

<table>
<thead>
<tr>
<th>Authentication</th>
<th>Authorization</th>
<th>Accounting</th>
</tr>
</thead>
<tbody>
<tr>
<td>• mysql -u admin</td>
<td>• Access Controls</td>
<td>• Logs</td>
</tr>
<tr>
<td></td>
<td>• -ppassword1</td>
<td>• Audit log</td>
</tr>
<tr>
<td>• Authentication Plugins</td>
<td>• Isolation</td>
<td>• General log</td>
</tr>
<tr>
<td>• SSL encryption</td>
<td></td>
<td>• Binary log</td>
</tr>
</tbody>
</table>
### MySQL Security

#### Authentication

- `mysql -u admin -ppassword1`
- Authentication Plugins
- SSL encryption
Authentication plugins

- MySQL pre-4.1 (old_password): DO NOT USE
- MySQL mysql_native_password: only use in exceptional cases
- MySQL caching_sha256 (since 8.0)
Why do we need to use caching_sha256?

mysql 5.7> create user b identified by 'password1';
mysql 5.7> create user c identified by 'password1';

mysql> select user, plugin, authentication_string from mysql.user
where user in ('a', 'b');

+--------+----------------+-------------------------------------------+
| user   | plugin         | authentication_string                  |
|--------+----------------+-------------------------------------------|
| a      | mysql_native_password | *668425423DB5193AF921380129F465A6425216D0 |
| b      | mysql_native_password | *668425423DB5193AF921380129F465A6425216D0 |
+--------+----------------+-------------------------------------------+
2 rows in set (0.00 sec)

Unsalted:
SAME hashes
Why do we need to use caching_sha256?

mysql 5.7> select password('password1');
+-------------------------------------------+
| password('password1')                     |
| *668425423DB5193AF921380129F465A6425216D0 |
+-------------------------------------------+

mysql 5.7> SELECT CONCAT('*', UPPER(SHA1(UNHEX(SHA1('password1')))));
+----------------------------------------------------+
| CONCAT('*', UPPER(SHA1(UNHEX(SHA1('password1'))))) |
| *668425423DB5193AF921380129F465A6425216D0          |
+----------------------------------------------------+
Why do we need to use caching_sha256?

mysql 8.0> create user a identified by 'password1';
mysql 8.0> create user b identified by 'password1';
mysql 8.0> select user, plugin, authentication_string from mysql.user
   where user in ('a','b');

<table>
<thead>
<tr>
<th>user</th>
<th>plugin</th>
<th>authentication_string</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>caching_sha2_password</td>
<td>$A$005$y!NOQ/9&lt;x}hZp5ffcvQ4sbcTpFkdf87jeWZSUdKLEftDe1vCK5BJWlp9</td>
</tr>
<tr>
<td>b</td>
<td>caching_sha2_password</td>
<td>$A$005$r*MAH&amp;4ZtC9sGJVm.6V/TJmLHgIIYbCLnXAkN2ZFJi82kPD3hiC</td>
</tr>
</tbody>
</table>

mysql 8.0> select user, plugin, hex(authentication_string) from mysql.user
   where user in ('a','b');

<table>
<thead>
<tr>
<th>user</th>
<th>plugin</th>
<th>hex(authentication_string)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>caching_sha2_password</td>
<td>244124303035240679214E1…</td>
</tr>
<tr>
<td>b</td>
<td>caching_sha2_password</td>
<td>24412430303524721A2A1A4…</td>
</tr>
</tbody>
</table>

salted – different hashes
But how did a bad actor get the password from hash?

1. Brutforce with hashcat (or other tools)
2. In some cases (able to sniff traffic and no SSL): can auth with hash: [https://github.com/cyrus-and/mysql-unsha1](https://github.com/cyrus-and/mysql-unsha1)
But how did a bad actor get the password from hash?

https://www.percona.com/blog/2020/06/12/brute-force-mysql-password-from-a-hash/
But how did a bad actor get the password from hash?

```
$ hashcat -m 300 -a 0 -D 2 -o -w 3 ./h ./rockyou.txt
```

Dictionary cache hit:
* Filename..: ./rockyou.txt
* Passwords.: 14344384
* Bytes.....: 139921497
* Keyspace..: 14344384

668425423db5193af921380129f465a6425216d0:password1

Session.........: hashcat
Status..........: Cracked
Hash.Name........: MySQL4.1/MySQL5
Hash.Target......: 668425423db5193af921380129f465a6425216d0
Time.Started....: Thu Jul 28 23:18:41 2022 (1 sec)
Time.Estimated...: Thu Jul 28 23:18:42 2022 (0 secs)
...

Started: Thu Jul 28 23:18:39 2022
Stopped: Thu Jul 28 23:18:43 2022
**But how did a bad actor get the password from hash?**

```sql
mysql 8.0> select authentication_string from mysql.user
where user = 'app_legacy'
+---------------------------------------------------------------------+
| authentication_string                                              |
+---------------------------------------------------------------------+
| *17C026786E36EE4E76098CC918AB00798DD0AA8C                          |
+---------------------------------------------------------------------+

hashcat# cat legacy
17C026786E36EE4E76098CC918AB00798DD0AA8C
mysql_native_password

Can we crack this?
But how did a bad actor get the password from hash?

```
17c026786e36ee4e76098cc918ab00798dd0aa8c:Q1b3-d
Session.........: hashcat
Status..........: Cracked
Time.Started....: Fri Jul 29 15:57:24 2022 (17 secs)
Hash.Name.......: MySQL4.1/MySQL5
Guess.Charset...: -1 ?l?u?d?s, -2 Undefined, -3 Undefined, -4 Undefined
Guess.Queue.....: 1/1 (100.00%)
Speed.#1........: 5884.2 MH/s (6.63ms) @ Accel:8
Recovered.......: 1/1 (100.00%) Digests
Progress.......: 95011471360/735091890625 (12.93%)
Rejected........: 0/95011471360 (0.00%)
Restore.Point...: 10485760/81450625 (12.87%)
Restore.Sub.#1.: Salt:0 Amplifier:512-576 Iteration:0
Candidate.Engine.: Device Generator
Candidates.#1...: CoS(KK -> ad"^~t
```

Cracked in 17 sec, brutforce mode, 6 random chars
But how did a bad actor get the password from hash?

mysql 8.0> SELECT
CONCAT('\\mysql',LEFT(authentication_string,6),'*',INSERT(HEX(SUBSTR(authentication_string
,8)),41,0,'*'))) AS hash FROM mysql.user WHERE plugin = 'caching_sha2_password' AND
user='app2';

$mysql$A$005*360907671C5A3E4A6D53564E47261E2F12562954*643143524E2F696A78534B684454
4F544A346D70743453664A4B71746F5075643339654F65664954544F44

caching_sha2_password

Can we crack this?
But how did a bad actor get the password from hash?

Session............: hashcat
Status.............: Running
Hash.Name..........: MySQL $A$ (sha256crypt)
Hash.Target........: $mysql$A$005*360907671C5A3E4A6D53564E47261E2F125629...544F44
Time.Started......: Fri Jul 29 16:09:17 2022 (1 min, 29 secs)
Time.Estimated.....: Mon Aug 29 14:23:15 2022 (30 days, 22 hours)
Kernel.Feature.....: Pure Kernel
Guess.Charset......: -1 ?l?u?d?s, -2 Undefined, -3 Undefined, -4 Undefined
Guess.Queue........: 1/1 (100.00%)
Speed.#1...........: 275.1 kH/s (7.38ms) @ Accel:8 Loops:16 Thr:1024 Vec:1
Recovered...........: 0/1 (0.00%) Digests
Progress...........: 24248320/735091890625 (0.00%)
Rejected............: 0/24248320 (0.00%)
Restore.Point......: 0/7737809375 (0.00%)
Restore.Sub.#1.....: Salt:0 Amplifier:37-38 Iteration:1920-1936
Candidate.Engine..: Device Generator
Candidates.#1......: Carier -> CYiQUS
But how did a bad actor get the password from hash?

8-chars password with lower and upper case letters and digits for MySQL 5.7 can be recovered only in 2 hours. The same password for MySQL 8.0 can be recovered in 2.8 years.

https://www.percona.com/blog/2020/06/12.brute-force-mysql-password-from-a-hash/
AuthN conclusion: use caching_sha2_password

1. Use caching_sha2_password
2. Use better passwords
3. Make sure the mysql.user is not easily readable
Our agenda

- **Encryption**
  - In flight (client->server)
  - At rest (on disk)

- **Authentication**
  - User/password

- **Authorization**
  - Grants/roles

- **Accounting**
  - Logs: audit log

We are here
MySQL Security

Authorization

- Access Controls
- Isolation
MySQL SQL Injection Example

```python
@app.route('/api/v1/resources/books')
def api_filter():
    query_parameters = request.args
    published = query_parameters.get('published')
    author = query_parameters.get('author')

    query = "SELECT * FROM books WHERE"
    if published:
        query += ' published=' + published
    ...
    cursor = mysql.connect().cursor()
    cursor.execute(query)
    results = cursor.fetchall()
    return jsonify(results)
```

*Adapted code / do not try at home*
MySQL SQL Injection Example

SELECT * FROM books WHERE published = 1
union
select user, host, authentication_string
from mysql.user

```
[
  "root",
  ":",
  "*E74858DB86EBA20BC33D0A",
  null,
  null
],
```

SQL injection!

But why the application user can select from mysql.user?
MySQL privilege model: History

Typical web application with MySQL database (since ~1996)
MySQL privilege model: History

Typical web applications with MySQL database (since ~1996)

# create superuser
GRANT ALL PRIVILEGES ON *.* TO 'root'@'\%'
WITH GRANT OPTION

Super user ("root" user) = can do everything

# create web app db and user
CREATE DATABASE WebApp;
GRANT ALL PRIVILEGES ON `WebApp`.* TO 'app'@'10.%'

# create backend db and user
CREATE DATABASE BackDB;
GRANT ALL PRIVILEGES ON `BackDB`.* TO 'cron'@'10.%'

"mysql" db is isolated
MySQL privilege model

Database per customer approach

mysql> show databases;
+-------------------------+
| Database                |
+-------------------------+
| sakila_00KBQa3SpbN9Brvv |
| sakila_014TcXtsuUviCaB7 |
| sakila_01nThhs7ch0l2aD7 |
| ...                    |
| sakila_zxwrBTtamcQBvbQ8 |
| sakila_zyp3nsGmUmBZPAQn |
| sakila_zzDtuzIyy7fHcLtT |
+-------------------------+
3075 rows in set (0.01 sec)

DB per customer

App creates DB

Good enough for DB isolation?
MySQL privilege model

Database per customer approach: Wrong way

```
GRANT SELECT, INSERT, UPDATE, DELETE
ON *.* TO app@'10.%';
```

What is the risk?

mysql> select user, host, authentication_string
       from mysql.user;

+------------------+-----------+-----------------------------------+
| user | host | authentication_string |
+------------------+-----------+-----------------------------------+
| admin | % | *C8307................|

App user can read hash of password for any user!
What is the risk?

```sql
mysql> show grants;
+---------------------------------------------------------------------+
|                  Grants for app@%                                    |
+---------------------------------------------------------------------+
| GRANT SELECT, INSERT, UPDATE, DELETE ON *.* TO 'app'@'%'            |
+---------------------------------------------------------------------+

mysql> update mysql.user set authentication_string = password('new_password')
    -> where user = 'admin';
Query OK, 1 row affected
...
$ mysql -uadmin -pnew_password

mysql> show grants;
+---------------------------------------------------------------------+
|                  Grants for admin@%                                   |
+---------------------------------------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'admin'@'%' WITH GRANT OPTION         |
+---------------------------------------------------------------------+
1 row in set (0.00 sec)
```
What is the risk?

mysql> show grants;
+---------------------------------------------+
| Grants for app@%                           |
| GRANT SELECT, INSERT, UPDATE, DELETE ON *.* TO 'app'@'%' |
| GRANT ALL PRIVILEGES ON *.* TO 'app'@'%'    |
+---------------------------------------------+
2 rows in set (0.00 sec)

mysql> update mysql.user set super_priv='y'
    -> where user='app';
Query OK, 1 row affected (0.00 sec)
...
mysql> show grants;
+---------------------------------------------+
| Grants for app@%                           |
| GRANT SELECT, INSERT, UPDATE, DELETE, SUPER ON *.* TO 'app'@'%' |
+---------------------------------------------+

Controlling mysql.user table means that you can get any additional privilege.
MySQL privilege model

Database per customer approach: Right way

```
# grant multiple database access
GRANT ALL PRIVILEGES
  ON `sakila_%`.* TO 'app'@'10.%'

mysql> create database aaa;
ERROR 1044 (42000): Access denied for user 'app'@'10.%' to database 'aaa'

mysql> create database sakila_test;
Query OK, 1 row affected (0.01 sec)
```

```
mysql> show databases;
+-------------------------------+
<table>
<thead>
<tr>
<th>Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>information_schema</td>
</tr>
<tr>
<td>sakila_00KBQa3SpbN9Brvv</td>
</tr>
<tr>
<td>sakila_014TcXtsuUviCaB7</td>
</tr>
<tr>
<td>sakila_01nThhs7ch0l2aD7</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>sakila_zxwrBTtamcQBvbQ8</td>
</tr>
<tr>
<td>sakila_zyp3nsGmUmBZPAQn</td>
</tr>
<tr>
<td>sakila_zzDtuzIyy7fHcLtT</td>
</tr>
</tbody>
</table>
+-------------------------------+
3075 rows in set (0.01 sec)
```
MySQL privilege model

Database per customer approach

What if there is no distinct pattern?

mysql> show databases;
+------------------+
| Database          |
+------------------+
| 00KBQa3SpbN9Brvv |
| 014TcXtsuUviCaB7 |
| 01nThhS7ch0l2aD7 |
| ...              |
| zxwrBTtamcQBvbQ8 |
| zyp3nsGmUmBZPAQn |
| zzDtuzIyy7fHcLtT |
+------------------+
3075 rows in set

Easy to make mistake and GRANT ALL PRIVILEGES to app user!
Easy to make mistake...

Summary

User with global read/write access

update mysql.user table

get any new DB privilege

Privilege escalation path
Changes in MySQL 8.0

New options

- Partial revokes
- Roles
- Dynamic privileges
MySQL 8.0 privilege model changes

Partial revokes: Game changer

“Prior to MySQL 8.0.16, it is not possible to grant privileges that apply globally except for certain schemas. As of MySQL 8.0.16, that is possible if the `partial_revokes` system variable is enabled.”


Exactly what we need
MySQL 8.0 privilege model changes

Before: MySQL 5.7

```
mysql 5.7> show grants for app@'10.%';
+-------------------------------------------------------------------+
| Grants for app@10.%                                               |
+-------------------------------------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'app'@'10.%'                       |
+-------------------------------------------------------------------+
1 row in set (0.00 sec)
```

```
mysql 5.7 > revoke all on mysql.* from app@'10.%';
ERROR 1141 (42000): There is no such grant defined for user 'app' on host '10.%'
```

```
mysql> show databases;
+------------------+
| Database         |
+------------------+
| 00KBQa3SpbN9Brvv |
| 014TcxtsuUviCaB7 |
| 01nThhs7ch0l2aD7 |
| ...              |
| mysql            |
+------------------+
3075 rows in set
```
MySQL 8.0 privilege model changes

Partial revokes in MySQL 8.0

mysql 8.0> SET global partial_revokes = ON;
Query OK, 0 rows affected (0.00 sec)

mysql 8.0> grant all on *.* to app@'10.%';
Query OK, 0 rows affected (0.01 sec)

mysql 8.0> revoke all on mysql.* from app@'10.%';
Query OK, 0 rows affected (0.01 sec)

mysql> show databases;
+------------------+
| Database         |
+------------------+
| 00KBQa3SpbN9Brvv |
| 014TcXtsuUviCaB7 |
| 01nThhS7ch0l2aD7 |
| ...              |
| mysql            |
| ...              |
| zxwrBTtamcQBvbQ8 |
| zyp3nsGmUmBZPAQn |
| zzDtuzIyy7fHcLtT |
+------------------+
3075 rows in set
MySQL 8.0 privilege model changes

Partial revokes in MySQL 8.0

# after revoke: access to any db will work
mysql 8.0> create table 00KBQa3SpbN9Brvv.a(i int);
Query OK, 0 rows affected (0.02 sec)
mysql 8.0> insert into 00KBQa3SpbN9Brvv.a values(1);
Query OK, 1 row affected (0.01 sec)

# access to mysql db will FAIL
mysql 8.0> update mysql.user
       set authentication_string = 'new'
       where user = 'root';
ERROR 1142 (42000): UPDATE command denied to user 'app'@'10.%' for table 'user'

mysql> show databases;
+------------------+
| Database         |
+------------------+
| 00KBQa3SpbN9Brvv |
| 014TcXtsuUviCaB7 |
| 01nThhS7ch0l2aD7 |
| ...              |
| mysql            |
| ...              |
| zxwrBTtamicQBvbQ8 |
| zyp3nsGmUmBZPAQn |
| zzDtuzIyy7fHcLtT |
+------------------+
3075 rows in set
MySQL 8.0 privilege model changes

Roles

“A MySQL role is a named collection of privileges. Like user accounts, roles can have privileges granted to and revoked from them.”

MySQL 8.0 privilege model changes

Convert app user to roles: Creating role

```sql
# create role
mysql 8.0> create role app_role;
Query OK, 0 rows affected (0.00 sec)

# grant privileges to ROLE
mysql 8.0> grant all on *.* to app_role;
Query OK, 0 rows affected (0.01 sec)

# revoke mysql db privileges from ROLE
mysql 8.0> revoke all on mysql.* from app_role;
Query OK, 0 rows affected (0.01 sec)
```

Easier to manage user permissions with roles
MySQL 8.0 privilege model changes

Convert app user to roles: Applying role to user

# cleanup all privileges first
mysql 8.0> revoke all on *.* from app;
Query OK, 0 rows affected (0.00 sec)

# assign role
mysql 8.0> grant app_role to app;
Query OK, 0 rows affected (0.01 sec)

# make it default
mysql 8.0> set default role app_role to app;
Query OK, 0 rows affected (0.00 sec)
MySQL 8.0 privilege model changes

Convert app user to roles: Applying role to user

# check grants
mysql 8.0> show grants for `app`@`%`;
+-----------------------------------------------------------+
| Grants for app@%                                         |
+-----------------------------------------------------------+
| GRANT USAGE ON `.*` TO `app`@`%`                          |
| GRANT `app_role`@`%` TO `app`@`%`                        |
+-----------------------------------------------------------+
2 rows in set (0.00 sec)

USAGE stands for “no privileges.” SHOW GRANTS displays USAGE to indicate that an account has no privileges at a privilege level.
MySQL 8.0 privilege model changes

Convert app user to roles: Applying role to user

```sql
# check grants
mysql 8.0> create table 00KBQa3SpbN9Brvv.b(i int);
Query OK, 0 rows affected (0.02 sec)

mysql 8.0> insert into 00KBQa3SpbN9Brvv.b values(1);
Query OK, 1 row affected (0.00 sec)

mysql 8.0> update mysql.user
    set authentication_string = 'new'
    where user = 'root';
ERROR 1142 (42000): UPDATE command denied to user 'app'@10.%' for table 'user'

mysql 8.0> revoke all on mysql.* from app_role;
```

```
mysql> show databases;
+------------------+
| Database         |
+------------------+
| mysql            |
| ...              |
3075 rows in set
```

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.
AuthZ: Security and Isolation

User with global read/write access

= update mysql.user table

Access denied ... to database 'mysql'
Our agenda

Encryption
- In flight (client->server)
- At rest (on disk)

Authentication
- User/password

Authorization
- Grants/roles

Accounting
- Logs: audit log

We are here
MySQL Security

Accounting

- Logs
  - Audit log
  - General log
  - Binary log
Why do we need audit log?

One of the use case: find attack and research who did it:

```xml
<AUDIT_RECORD
  "NAME"="Query"
  "TIMESTAMP"="2022-04-29T10:20:10 UTC"
  "COMMAND_CLASS"="select"
  "CONNECTION_ID"="49"
  "STATUS"="0"
  "SQLTEXT"=""
SELECT * FROM books WHERE published = 1
union
select user, host, authentication_string
from mysql.user"
  "USER"="app[app] @ 10.0.0.2 []"
  "HOST"="10.0.0.2"
/>```

SQL injection

User / host

Timestamp
I am building an new team in Amazon Web Services RDS and looking for Security / Red Team / Pentest Engineers.


Job details: