Encrypting binary (and relay) logs in MySQL

Matthias Crauwels

FOSDEM 2022 - Online

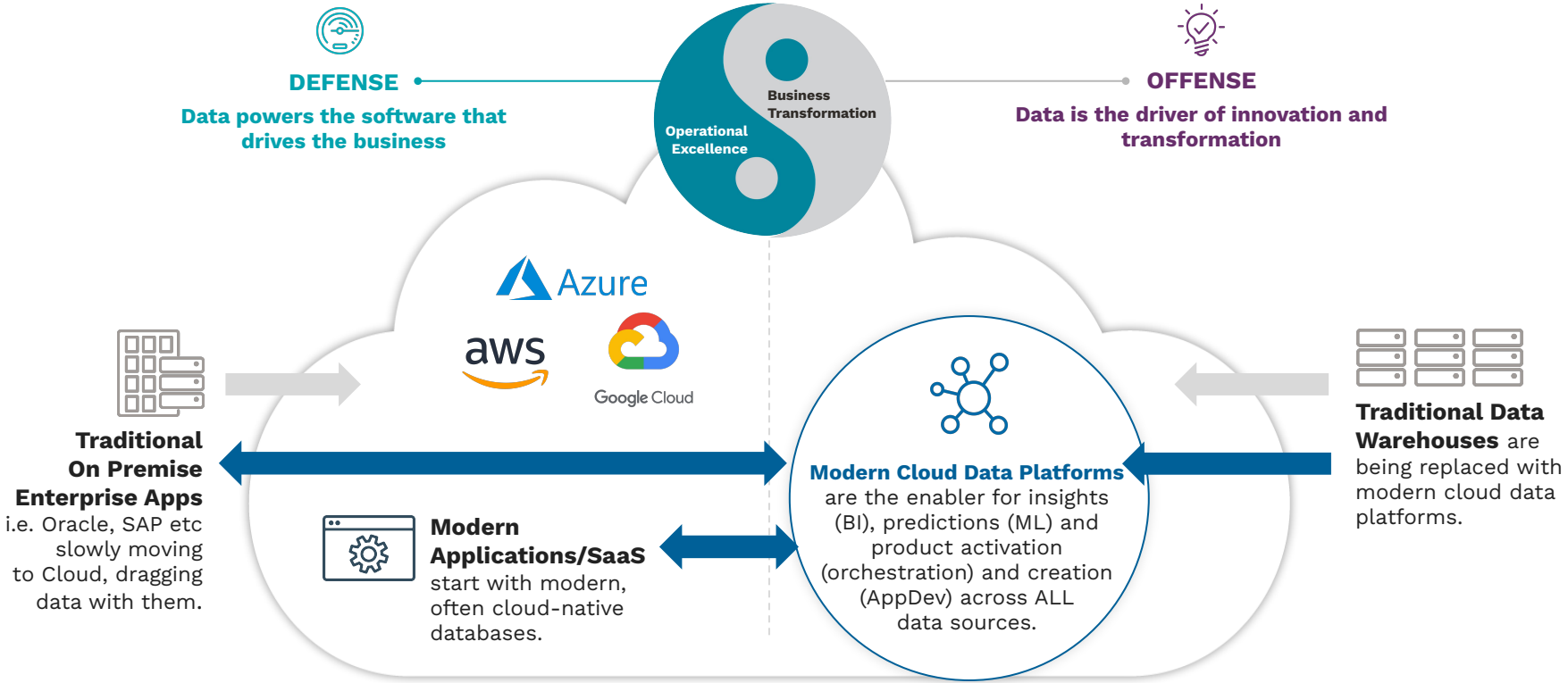
Sat Feb 5th

Pythian

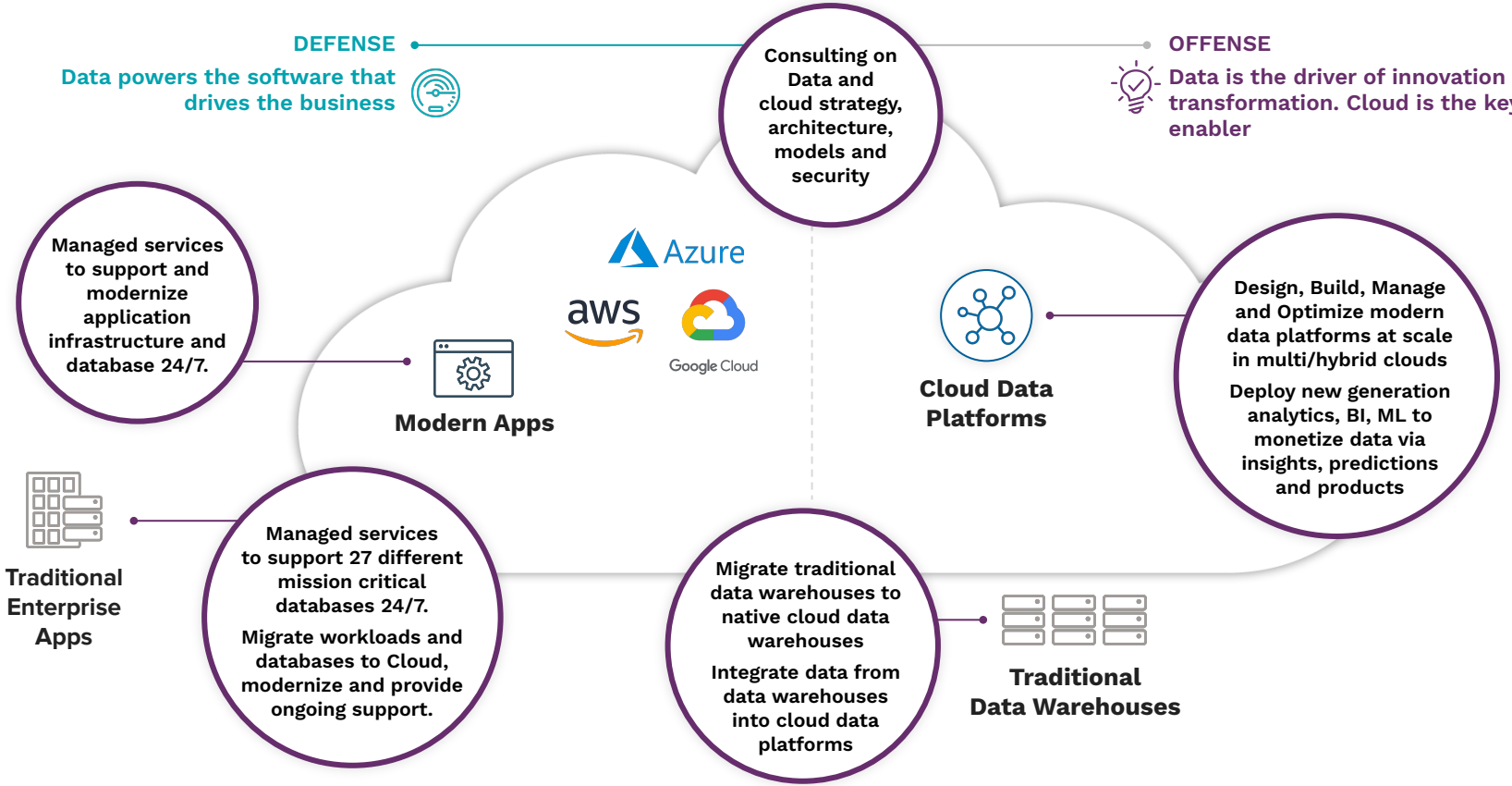
Speaker

Matthias Crauwels
Principal Consultant
Pythian - OSDB

How the data estate is evolving



Pythian's Services Across the Data Estate



AGENDA

- Introduction to MySQL Security features
- Encrypting the binary logs
- Keyring plugins
- Decrypting a binary log file

Default MySQL Security features (as of 5.7)

- MySQL generates a secure root password

```
[root@localhost ~]# systemctl start mysqld
[root@localhost ~]# cat /var/log/mysqld.log | grep 'temporary password'
2022-01-22T09:14:51.074966Z 6 [Note] [MY-010454] [Server] A temporary password is generated for
root@localhost: oDuMK*ey!3u(
```

- The root-account is locked

```
mysql> SELECT * FROM mysql.user;
ERROR 1820 (HY000): You must reset your password using ALTER USER statement before executing this
statement.
```

- The `validate_password` plugin/component is enabled
- SSL certificates are generated and used for TCP/IP connections

MySQL + encryption in flight

- SSL connectivity is using self-signed certificate
- Connection is encrypted but the identity of the server can not be verified
- Best practice would be to use your company's Certification Authority to sign a valid certificate so the identity could be verified.

```
[root@localhost mysql]# openssl x509 -in /var/lib/mysql/server-cert.pem -noout -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 2 (0x2)
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN = MySQL_Server_8.0.28_Auto_Generated_CA_Certificate
        Validity
            Not Before: Jan 22 09:14:50 2022 GMT
            Not After : Jan 20 09:14:50 2032 GMT
        Subject: CN = MySQL_Server_8.0.28_Auto_Generated_Server_Certificate
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
```

MySQL + encryption at rest

3 major options

- Encryption at disk level
- Encryption at database/table level
- Encryption at application level

MySQL: encryption at disk level

- Encryption is done at the OS level
- Protects you against someone pulling out a disk from the server
- All or nothing encryption
- Once you get into the server you can still copy the files to a non-encrypted volume and get away with the data anyway

MySQL: encryption at database/table level

- Available since 5.7
- Only available for InnoDB:

```
mysql> CREATE TABLE t1 (c1 INT) ENGINE=InnoDB ENCRYPTION='Y';
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE TABLE t2 (c1 INT) ENGINE=MyISAM ENCRYPTION='Y';
ERROR 1178 (42000): The storage engine for the table doesn't support ENCRYPTION
```

- Since 8.0.16 there is an option to enable table encryption by default.

```
SET GLOBAL default_table_encryption=ON;
```


MySQL: encryption at application level

- Most granular type of encryption
- At the discretion of the developer
- Data is encrypted BEFORE it's stored in the MySQL server
- Only the application logic knows which data was encrypted and how to decrypt it

Beyond table-data encryption

- Doublewrite file encryption (since 8.0.23)

  Automatically enabled for encrypted tablespaces
- mysql system tablespace encryption (since 8.0.16)

```
ALTER TABLESPACE mysql ENCRYPTION = 'Y';
```

- Redo and undo log encryption (since 8.0.1)

```
mysql> SET GLOBAL innodb_undo_log_encrypt = ON;
Query OK, 0 rows affected (0.00 sec)

mysql> SET GLOBAL innodb_redo_log_encrypt = ON;
Query OK, 0 rows affected (0.00 sec)
```

- Binary log encryption (since 8.0.14)

```
mysql> SET GLOBAL binlog_encryption = ON;
Query OK, 0 rows affected (0.02 sec)
```


MySQL binary log encryption: How?

```
mysql> SET GLOBAL binlog_encryption = ON;

ERROR 3794 (HY000): Unable to recover binlog encryption master key, please
check if keyring is loaded.
```

For all of the encryption features that MySQL supports you will need to load a keyring plugin or component.

MySQL keyring plugins

- **MySQL Community Edition** comes with one keyring plugin:
  - `keyring_file` Stores keyring data in a file local to the server host
- **MySQL Enterprise Edition** comes with more plugins:
  - `keyring_encrypted_file`: Similar to `keyring_file` but encrypt and password protect the file
  - `keyring_okv`: plugin to use with Oracle Key Vault
  - `keyring_aws`: plugin to use AWS Key Management Service
  - `keyring_hashicorp`: plugin to use Hashicorp Vault
  - `keyring_oci`: plugin to use Oracle Cloud Infrastructure Vault
- **Percona Server** adds an open source plugin to use Hashicorp Vault

MySQL keyring plugins: How?

In your `my.cnf` add these lines in the `[mysqld]` section

```
early-plugin-load=keyring_file.so
keyring-file-data=/var/lib/mysql-keyring/keyring
```

Restart MySQL

```
[root@localhost ~]# systemctl restart mysqld
[root@localhost ~]# mysql
...
mysql> SHOW PLUGINS;
+---------------------------------+----------+--------------------+-----------------+---------+
| Name                            | Status   | Type               | Library         | License |
+---------------------------------+----------+--------------------+-----------------+---------+
...
| keyring_file                    | ACTIVE   | KEYRING            | keyring_file.so | GPL     |
...
+---------------------------------+----------+--------------------+-----------------+---------+
```


MySQL binary log encryption: How?

```
mysql> SET GLOBAL binlog_encryption = ON;
Query OK, 0 rows affected (0.02 sec)
```

Great success!

```
mysql> SHOW BINARY LOGS;
+----------------------+-----------+-----------+
| Log_name             | File_size | Encrypted |
+----------------------+-----------+-----------+
| localhost-bin.000001 |       180 | No        |
| localhost-bin.000002 |       501 | No        |
| localhost-bin.000003 |       248 | No        |
| localhost-bin.000004 |      1083 |  Yes      |
+----------------------+-----------+-----------+
4 rows in set (0.00 sec)
```

If you enable `binlog_encryption`, this server will also automatically encrypt any relay logs that it writes. **So don't forget to enable `binlog_encryption` on all your replica's**

MySQL binary log encryption: Now what?

```
[root@localhost mysql]# mysqlbinlog localhost-bin.000004
# The proper term is pseudo_replica_mode, but we use this compatibility
alias
# to make the statement usable on server versions 8.0.24 and older.
/*!50530 SET @@SESSION.PSEUDO_SLAVE_MODE=1*/;
/*!50003 SET @OLD_COMPLETION_TYPE=@@COMPLETION_TYPE,COMPLETION_TYPE=0*/;
DELIMITER /*!*/;
ERROR: Reading encrypted log files directly is not supported.
SET @@SESSION.GTID_NEXT= 'AUTOMATIC' /* added by mysqlbinlog */ /*!*/;
DELIMITER ;
# End of log file
/*!50003 SET COMPLETION_TYPE=@OLD_COMPLETION_TYPE*/;
/*!50530 SET @@SESSION.PSEUDO_SLAVE_MODE=0*/;
```

MySQL binary log encryption: Now what?

- How can I use the binary logs to find my transactions?
- How can I do point-in-time recovery using the binary logs?

Some Google-fu landed me on this blog post:

https://dev.mysql.com/blog-archive/how-to-manually-decrypt-an-encrypted-binary-log-file/

MySQL engineer João Gramacho explains in great detail how the encryption is done and he also provides a shell script to decrypt the binary log files.

Decrypting the binary log files

First let's make sure that the binary log file is not being used anymore

```
mysql> FLUSH BINARY LOGS;
Query OK, 0 rows affected (0.01 sec)

mysql> SHOW BINARY LOGS;
+----------------------+-----------+-----------+
| Log_name             | File_size | Encrypted |
+----------------------+-----------+-----------+
| localhost-bin.000001 |       180 | No        |
| localhost-bin.000002 |       501 | No        |
| localhost-bin.000003 |       248 | No        |
| localhost-bin.000004 |      2758 | Yes       |
| localhost-bin.000005 |       709 | Yes       |
+----------------------+-----------+-----------+
5 rows in set (0.00 sec)
```

Decrypting the binary log files

```
[root@localhost ~]# cp /var/lib/mysql/localhost-bin.000004 .
[root@localhost ~]# ls -hl
total 8.0K
-rwxr-xr-x. 1 root root 3.9K Jan 22 11:25 decrypt_binlog.sh
-rw-r-----. 1 root root 2.7K Jan 22 11:28 localhost-bin.000004
[root@localhost ~]#
```

I've copied my binary log to my working directory. And I've downloaded João's script also to my working directory.

Decrypting the binary log files

```
[root@localhost ~]#  ./decrypt_binlog.sh
Error: Please specify the binary log file to decrypt.

Usage: decrypt_binlog.sh <BINARY LOG FILE> [<KEYRING KEY VALUE>]
Where:
      <BINARY LOG FILE>:
          The binary or relay log file to be decrypted.
      <KEYRING KEY VALUE>:
          The keyring key value to decrypt the file.
          It shall be passed in hexadecimal notation.
          If not specified, the program will display the key ID that.
          is required to decrypt the file.

[root@localhost ~]# ./decrypt_binlog.sh localhost-bin.000004
Keyring key ID for 'localhost-bin.000004' is
'MySQLReplicationKey_34b46de1-7b6e-11ec-a7ee-080027fce996_1'
```

Okay? So where do I get this key from?

Decrypting the binary log files

```
mysql> SELECT * FROM performance_schema.keyring_keys;
+-------------------------------------------------------------------+-----------+----------------+
| KEY_ID                                                            | KEY_OWNER | BACKEND_KEY_ID |
+-------------------------------------------------------------------+-----------+----------------+
| MySQLReplicationKey_34b46de1-7b6e-11ec-a7ee-080027fce996_1        |           |                |
| MySQLReplicationKey_34b46de1-7b6e-11ec-a7ee-080027fce996          |           |                |
+-------------------------------------------------------------------+-----------+----------------+
2 rows in set (0.00 sec)
```

Great! The key is actually in my keyring, now how do I get it out?

Decrypting the binary log files

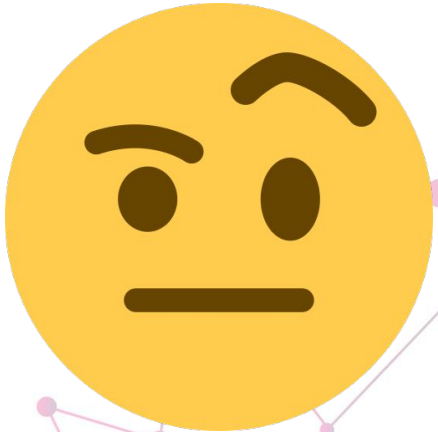
MySQL provides some general purpose keyring function as user-defined functions (UDF). The reference manual has instructions on how to install these.

```
INSTALL PLUGIN keyring_udf SONAME 'keyring_udf.so';
CREATE FUNCTION keyring_key_generate RETURNS INTEGER
  SONAME 'keyring_udf.so';
CREATE FUNCTION keyring_key_fetch RETURNS STRING
  SONAME 'keyring_udf.so';
CREATE FUNCTION keyring_key_length_fetch RETURNS INTEGER
  SONAME 'keyring_udf.so';
CREATE FUNCTION keyring_key_type_fetch RETURNS STRING
  SONAME 'keyring_udf.so';
CREATE FUNCTION keyring_key_store RETURNS INTEGER
  SONAME 'keyring_udf.so';
CREATE FUNCTION keyring_key_remove RETURNS INTEGER
  SONAME 'keyring_udf.so';
```

`keyring_key_fetch` seems like a good candidate. Let's give it a try.

Decrypting the binary log files

```
mysql> SELECT keyring_key_fetch('MySQLReplicationKey_34b46de1-7b6e-11ec-a7ee-080027fce996_1') as
encryption_key;
+---------------------------------+
| encryption_key                  |
+---------------------------------+
| NULL                            |
+---------------------------------+
1 row in set (0.00 sec)
```



Decrypting the binary log files

Manually creating a key in the keyring

```
mysql> SELECT keyring_key_generate('MyKey', 'RSA', 64) as encryption_key;
+----------------+
| encryption_key |
+----------------+
|              1 |
+----------------+

mysql> SELECT * FROM performance_schema.keyring_keys;
+-------------------------------------------------------------+----------------+-----------------+
| KEY_ID                                                      | KEY_OWNER      | BACKEND_KEY_ID |
+-------------------------------------------------------------+----------------+-----------------+
...
| MyKey                                                       | root@localhost |                 |
+-------------------------------------------------------------+----------------+-----------------+

mysql> SELECT keyring_key_fetch('MyKey') as encryption_key;
+-------------------------------------------------------------------------------------------------+
| encryption_key                                                                                  |
+-------------------------------------------------------------------------------------------------+
| 0xF7DD1291C1C229D77080838D4648DC7A...9E1C62BC46EA292FC9BBC47C9DBCF2249EE57ACC5B6700AE08FF50A |
+-------------------------------------------------------------------------------------------------+
```

Decrypting the binary log files

New approach let's have a look at the keyring itself. I used the `keyring_file` plugin for this example, storing the keyring in file on the system in `/var/lib/mysql-keyring/keyring`

```
[root@localhost ~]# ls -hl /var/lib/mysql-keyring/keyring
-rw-r-----. 1 mysql mysql 443 Jan 22 11:44 /var/lib/mysql-keyring/keyring
```

Let's copy this file also to our working directory to assess it.

Decrypting the binary log files

The keyring file is a binary file, so you can't just read it's contents, although you can make something out of it...

```
[root@localhost ~]# cat keyring
Keyring file version:2.0@MyKeyRSAroot@localhost""C@b
`.g~d□F□D□□□□}
t=QJŨgx8MySQLReplicationKey_34b46de1-7b6e-11ec-a7ee-080027fce996AES+305=Ljt0*!@$H
MySQLReplicationKey_34b46de1-7b6e-11ec-a7ee-080027fce996_1Af\?D$%d+T
-EOFcb!?Lu(u@m
[root@localhost ~]#
```

João's script to the rescue?

```
[root@localhost ~]# ./decrypt_binlog.sh localhost-bin.000004 keyring
hex string is too short, padding with zero bytes to length
non-hex digit
invalid hex key value
[root@localhost ~]#
```

Decrypting the binary log files

I **need** this binary log for my point-in-time recovery!

My Google-fu to the rescue!

I found another blog post, by Jesper Krogh, linking to João's original post. Jesper took João's script one step further and implemented in Python a script where you **can** specify the keyring file as a parameter to decrypt the binlog file.

https://mysql.wisborg.dk/2019/01/28/automatic-decryption-of-mysql-binary-logs-using-python/

Decrypting the binary log files

I downloaded Jesper's script to my working directory and installed the dependencies as he described them in his blog post.

```
[root@localhost ~]# ls -hl
total 24K
-rwxr-xr-x. 1 root root  12K Jan 22 12:05  decrypt_binlog.py
-rwxr-xr-x. 1 root root 3.9K Jan 22 11:25 decrypt_binlog.sh
-rw-r-----. 1 root root  443 Jan 22 11:50 keyring
-rw-r-----. 1 root root 2.7K Jan 22 11:28 localhost-bin.000004
```


Decrypting the binary log files

```
[root@localhost ~]# python3.6 decrypt_binlog.py -k keyring localhost-bin.000004
localhost-bin.000004: Keyring key ID for is
'MySQLReplicationKey_34b46de1-7b6e-11ec-a7ee-080027fce996_1'
localhost-bin.000004: Successfully decrypted as '/root/plain-localhost-bin.000004'
[root@localhost ~]# ls -hl
total 28K
-rwxr-xr-x. 1 root root  12K Jan 22 12:05 decrypt_binlog.py
-rwxr-xr-x. 1 root root 3.9K Jan 22 11:25 decrypt_binlog.sh
-rw-r-----. 1 root root  443 Jan 22 11:50 keyring
-rw-r-----. 1 root root 2.7K Jan 22 11:28 localhost-bin.000004
-rw-r--r--. 1 root root 2.2K Jan 22 12:08 plain-localhost-bin.000004
[root@localhost ~]#
```

Great success?!

Decrypting the binary log files

```
[root@localhost ~]# mysqlbinlog plain-localhost-bin.000004
...
#220122 10:57:39 server id 1  end_log_pos 382 CRC32 0x6d429f13  Query  thread_id=8  exec_time=0  error_code=0  Xid = 5
SET TIMESTAMP=1642849059/*!*/;
SET @@session.pseudo_thread_id=8/*!*/;
SET @@session.foreign_key_checks=1, @@session.sql_auto_is_null=0, @@session.unique_checks=1, @@session.autocommit=1/*!*/;
SET @@session.sql_mode=1168113696/*!*/;
SET @@session.auto_increment_increment=1, @@session.auto_increment_offset=1/*!*/;
/*!\C utf8mb4 *//*!*/;
SET @@session.character_set_client=255,@@session.collation_connection=255,@@session.collation_server=255/*!*/;
SET @@session.lc_time_names=0/*!*/;
SET @@session.collation_database=DEFAULT/*!*/;
/*!80011 SET @@session.default_collation_for_utf8mb4=255*//*!*/;
/*!80016 SET @@session.default_table_encryption=0*//*!*/;
CREATE DATABASE test
/*!*/;
# at 382
...
```

Decrypting the binary log files

Important to note is that Jesper's script only works for the keyring_file plugin. Quoting the Jesper's blog:

"The keyring must be from the keyring_file plugin and using format version 2.0 (the format current as of MySQL 8.0.14). If you use a different keyring plugin, you can use the keyring migration feature to create a copy of the keyring using keyring_file. (But, please note that keyring_file is not a secure keyring format.)"

MySQL keyring migration feature

If you want to use encrypted binary logs you probably don't want to use the `keyring_file` plugin as it not secure. Without specifying any password I could eventually decrypt my binary log file.

Let me try MySQL Enterprise `keyring_encrypted_file` plugin.

MySQL keyring migration feature

Enabling the keyring_encrypted_file plugin takes a parameter to store the keyring and a password to encrypt the data in the keyring

```
[root@node1 ~]# cat /etc/my.cnf | grep keyring
early-plugin-load=keyring_encrypted_file.so
keyring_encrypted_file_data=/var/lib/mysql-keyring/keyring-encrypted
keyring_encrypted_file_password=password
```


MySQL keyring migration feature

Binary log encryption is active

```
mysql> SHOW BINARY LOGS;
+----------------------+-----------+-----------+
| Log_name             | File_size | Encrypted |
+----------------------+-----------+-----------+
| localhost-bin.000001 |       179 | No        |
| localhost-bin.000002 |       498 | No        |
| localhost-bin.000003 |       219 | No        |
| localhost-bin.000004 |       247 | No        |
| localhost-bin.000005 |       759 | Yes       |
| localhost-bin.000006 |       944 | Yes       |
| localhost-bin.000007 |      2107 | Yes       |
| localhost-bin.000008 |       708 | Yes       |
+----------------------+-----------+-----------+
```


MySQL keyring migration feature

Preparing the keyring-migration config file

```
[root@node1 ~]# cat keyring-migration.cnf
[mysqld]
user=mysql

keyring_encrypted_file_data=/tmp/keyring-encrypted
keyring_file_data=/tmp/keyring
```

Checking if the encrypted keyring file is in place

```
[root@node1 ~]# ls -hl /tmp/key*
-rw-r-----. 1 mysql mysql 437 Jan 22 19:00 /tmp/keyring-encrypted
```


MySQL keyring migration feature

Running mysqld as the keyring migration service

```
[root@node1 ~]# mysqld --defaults-file=keyring-migration.cnf \
--keyring-migration-source=keyring_encrypted_file.so  \
--keyring-migration-destination=keyring_file.so  \
--keyring_encrypted_file_password=password
2022-01-22T19:04:53.688417Z 0 [System] [MY-010116] [Server] /usr/sbin/mysqld
(mysqld 8.0.22-commercial) starting as process 4755
2022-01-22T19:04:53.843074Z 0 [System] [MY-010910] [Server]
/usr/sbin/mysqld: Shutdown complete (mysqld 8.0.22-commercial)  MySQL
Enterprise Server - Commercial.
```


MySQL keyring migration feature

Verification if the decrypted file is actually there

```
[root@node1 ~]# ls -hl /tmp/key*
-rw-r-----. 1 mysql mysql 395 Jan 22 19:04 /tmp/keyring
-rw-r-----. 1 mysql mysql 437 Jan 22 19:00 /tmp/keyring-encrypted
```

MySQL keyring migration feature

With the encrypted keyring Jesper's python script fails

```
[root@node1 ~]# python3.6 binlog_decrypt.py -k /tmp/keyring-encrypted node1-bin.000004
Traceback (most recent call last):
  File "binlog_decrypt.py", line 301, in <module>
    main(sys.argv[1:])
  File "binlog_decrypt.py", line 297, in main
    decrypt_binlogs(args)
  File "binlog_decrypt.py", line 242, in decrypt_binlogs
    keyring = Keyring(args.keyring_file_data)
  File "binlog_decrypt.py", line 48, in __init__
    self.read_keyring(keyring_filepath)
  File "binlog_decrypt.py", line 88, in read_keyring
    .format(header.hex()))
ValueError: Invalid header in the keyring file: 4b657972696e6720656e63727970746564206669 6c
```

MySQL keyring migration feature

With the decrypted keyring file the script could successfully decrypt the binary log file

```
[root@node1 ~]# python3.6 binlog_decrypt.py -k /tmp/keyring node1-bin.000004
node1-bin.000004: Keyring key ID for is
'MySQLReplicationKey_e30eac4c-633c-11ec-92fc-5254008afee6_1'
node1-bin.000004: Successfully decrypted as '/root/plain-node1-bin.000004'
[root@node1 ~]#
```

Conclusion

Conclusion

- Encrypting binary logs is not hard
- Selecting a secure keyring is harder
  - The only secure open source keyring is Percona's `keyring_vault` plugin which requires you to have an Hashicorp Vault installation.
- When backing up binary logs for point-in-time recovery you will need to ensure that you also backup your keyring to be able to decrypt the binary logs when you need them
- `mysqld` can be used as a keyring-migration-tool
- Add Jesper's python script to your DBA toolbox

Thank You

email: crauwels@pythian.com
twitter: @mcrauwel

MySQL keyring migration feature

I created a vault server for the purpose of this demo

```
[root@localhost ~]# vault status
Key             Value
---             -----
Seal Type       shamir
Initialized     true
Sealed          false
Total Shares    1
Threshold       1
Version         1.9.2
Storage Type    file
Cluster Name    vault-cluster-929598a1
Cluster ID      1ca9171c-247d-1778-b6db-3570179e8fcc
HA Enabled      false
```

MySQL keyring migration feature

And I did a default configuration of my `keyring_vault` plugin

```
[root@localhost ~]# cat /etc/my.cnf | grep keyring
early-plugin-load="keyring_vault=keyring_vault.so"
loose-keyring_vault_config="/etc/my.cnf.d/keyring_vault.conf"

[root@localhost ~]# cat /etc/my.cnf.d/keyring_vault.conf
vault_url = https://127.0.0.1:8200
secret_mount_point = secret/mysql1
secret_mount_point_version = AUTO
token = s.DQkShRUw9B8y3eI6IxrCJyEh
vault_ca = /etc/sslkeys/vault.crt
```