

Hardened MySQL Deployment: Encrypted Communication, Access Control, and Backup Strategy

MYSQL HARDENING VIA TLS, ACCESS RESTRICTION, AND
ENCRYPTED BACKUPS

TEBOGO MATSEDING

1st Login

Installing MySQL Secure Installation

Create a Certificate Authority (CA)

Create the MySQL Server Certificate (Step 1)

Create the MySQL Server Certificate (Step 2)

Create the MySQL Server Certificate (Step 3)

Create the MySQL Server Certificate (Step 4)

Verify SSL/TLS certs

Configure MySQL to Use SSL

Restart MySQL

Verify SSL is Active

Test Remote Admin can connect via SSL Encryption

Installing Fail2Ban

Edit Local Jail Config File

Verify Jail is Active

Installed Auditd

Auditd Status

Track writes/changes to MySQL directory

Log every sudo command run

Log Review Using ausearch

Creating New Server for Backups

Installing MySQL-client on Backup Server

Added Backup Server to the UFW

Create a MySQL user for Backup Server

Granting Backup Server Privileges

Backup Server Connecting to MySQL

Create mysqldump script

Test Script

Set crontab -e to automate backup

Crontab -l to verify it has been set

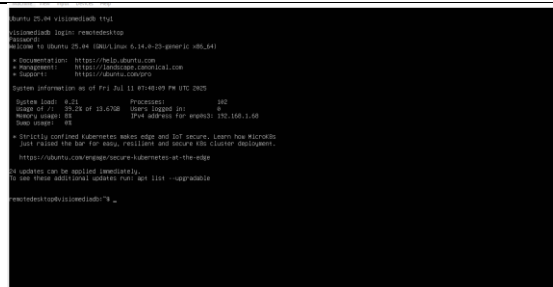
START

***NOTE**

This is a continuation from my previous project **“Secure Headless Database Infrastructure Using VirtualBox”** if you haven’t seen it, please click the link on the right of the table and it will direct you straight there, however if you have already seen it lets continue on

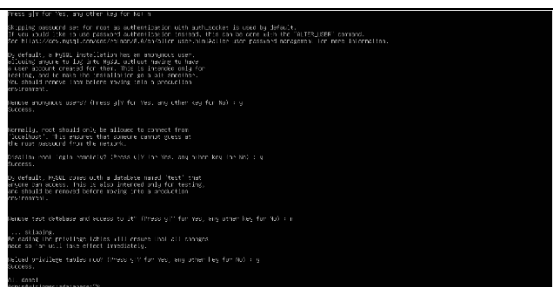
1st Login

Started by logging into the Remote Admin Server via SSH. This server acts as the secure entry point into the internal network and is the only machine allowed to communicate directly with the MySQL database server. From here, all configuration and monitoring tasks were performed.



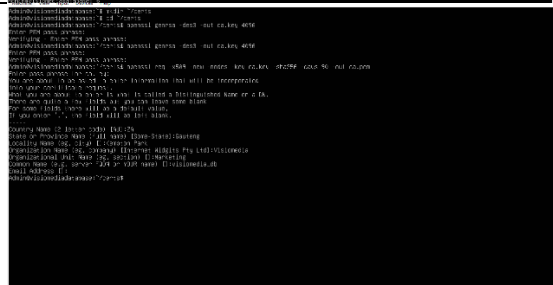
Installing MySQL Secure Installation

After installing MySQL, I ran the secure installation wizard. It's a good way to clean up the default settings I removed anonymous users, disabled root access over the network. Making sure MySQL didn't come with any open doors.



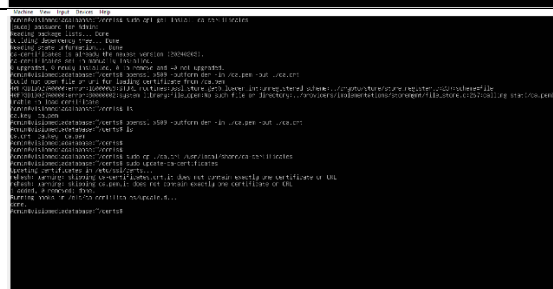
Create a Certificate Authority (CA)

Before enabling SSL, I needed a Certificate Authority. This is what signs all the other certificates in the setup it is kind of like creating my own internal trust system.



Create the MySQL Server Certificate

Generated a private key for the MySQL server. This key stays on the server and identifies it securely during SSL handshakes.



<p>Create the MySQL Server Certificate step 2</p> <p>Created a certificate signing request (CSR) based on the private key. This file is basically a formal request to the CA to get a proper certificate for the server.</p>	
<p>Create the MySQL Server Certificate step 3</p> <p>Signed the CSR using my CA to produce the actual server certificate. This is what MySQL uses to prove its identity to clients.</p>	
<p>Create the MySQL Server Certificate step 4</p> <p>Locked down the file permissions on all certificate files, then placed them in a secure location where MySQL could access them without leaving them exposed.</p>	
<p>Verify SSL/TLS certs</p> <p>Double-checked that the certificate chain was valid and all the files were working together properly, just to verifying everything was trusted and not expired or mismatched.</p>	
<p>Configure MySQL to Use SSL</p> <p>Added the certificate paths to MySQL's config file so it knows where to look when SSL connections are requested. This basically tells MySQL to enforce secure communication.</p>	
<p>Restart MySQL</p> <p>Restarted the MySQL service to load all the new SSL settings.</p>	

Connected to the MySQL server and confirmed that SSL was active by checking the variables. Everything showed up as expected, so encryption was working.

```

# Create a new table with 100 rows and 5 columns
df = pd.DataFrame({
    'id': range(100),
    'name': ['John', 'Jane', 'Bob', 'Alice', 'Charlie'] * 20,
    'age': range(20, 40),
    'gender': ['M', 'F'] * 50,
    'salary': range(30000, 70000)
})

# Display the first 10 rows
df.head(10)

# Filter rows where age is greater than 30
df[df['age'] > 30]

# Group by gender and calculate the average salary
df.groupby('gender')['salary'].mean()

# Sort by salary in descending order
df.sort_values('salary', ascending=False)

# Create a new column 'bonus' which is 10% of salary
df['bonus'] = df['salary'] * 0.1

# Drop the 'bonus' column
df.drop('bonus', axis=1)

# Save the modified DataFrame to a new CSV file
df.to_csv('modified_data.csv', index=False)

# Read the modified CSV file back into a DataFrame
df_modified = pd.read_csv('modified_data.csv')

# Verify the modification
df_modified.head(10)

```

Ran a test from a remote machine using the right SSL certificates. The goal here was to make sure that no one can connect without valid encryption the connection worked only when all certs were correct.

[illegible]

Installed Fail2Ban to protect against brute-force attacks. It reads log files and blocks IPs that fail too many times. Simple but super effective.

```

10: # Import the modules that we need
11: import pandas as pd
12: import numpy as np
13: from sklearn.metrics import confusion_matrix, classification_report
14: from sklearn.preprocessing import StandardScaler
15: from sklearn.model_selection import train_test_split
16: from sklearn.linear_model import LogisticRegression
17: from sklearn.svm import SVC
18: from sklearn.tree import DecisionTreeClassifier
19: from sklearn.ensemble import RandomForestClassifier
20: from sklearn.neural_network import MLPClassifier
21: from sklearn.metrics import accuracy_score
22: from sklearn.metrics import roc_auc_score
23: from sklearn.metrics import f1_score
24: from sklearn.metrics import precision_score
25: from sklearn.metrics import recall_score
26: from sklearn.metrics import log_loss
27: from sklearn.metrics import brier_score_function
28: from sklearn.metrics import jaccard_index_score
29: from sklearn.metrics import hamming_loss
30: from sklearn.metrics import cohen_kappa_score
31: from sklearn.metrics import matthews_correlation_coef
32: from sklearn.metrics import average_precision_score
33: from sklearn.metrics import average_precision_curve
34: from sklearn.metrics import average_precision_curve
35: from sklearn.metrics import average_precision_curve
36: from sklearn.metrics import average_precision_curve
37: from sklearn.metrics import average_precision_curve
38: from sklearn.metrics import average_precision_curve
39: from sklearn.metrics import average_precision_curve
40: from sklearn.metrics import average_precision_curve
41: from sklearn.metrics import average_precision_curve
42: from sklearn.metrics import average_precision_curve
43: from sklearn.metrics import average_precision_curve
44: from sklearn.metrics import average_precision_curve
45: from sklearn.metrics import average_precision_curve
46: from sklearn.metrics import average_precision_curve
47: from sklearn.metrics import average_precision_curve
48: from sklearn.metrics import average_precision_curve
49: from sklearn.metrics import average_precision_curve
50: from sklearn.metrics import average_precision_curve
51: from sklearn.metrics import average_precision_curve
52: from sklearn.metrics import average_precision_curve
53: from sklearn.metrics import average_precision_curve
54: from sklearn.metrics import average_precision_curve
55: from sklearn.metrics import average_precision_curve
56: from sklearn.metrics import average_precision_curve
57: from sklearn.metrics import average_precision_curve
58: from sklearn.metrics import average_precision_curve
59: from sklearn.metrics import average_precision_curve
60: from sklearn.metrics import average_precision_curve
61: from sklearn.metrics import average_precision_curve
62: from sklearn.metrics import average_precision_curve
63: from sklearn.metrics import average_precision_curve
64: from sklearn.metrics import average_precision_curve
65: from sklearn.metrics import average_precision_curve
66: from sklearn.metrics import average_precision_curve
67: from sklearn.metrics import average_precision_curve
68: from sklearn.metrics import average_precision_curve
69: from sklearn.metrics import average_precision_curve
70: from sklearn.metrics import average_precision_curve
71: from sklearn.metrics import average_precision_curve
72: from sklearn.metrics import average_precision_curve
73: from sklearn.metrics import average_precision_curve
74: from sklearn.metrics import average_precision_curve
75: from sklearn.metrics import average_precision_curve
76: from sklearn.metrics import average_precision_curve
77: from sklearn.metrics import average_precision_curve
78: from sklearn.metrics import average_precision_curve
79: from sklearn.metrics import average_precision_curve
80: from sklearn.metrics import average_precision_curve
81: from sklearn.metrics import average_precision_curve
82: from sklearn.metrics import average_precision_curve
83: from sklearn.metrics import average_precision_curve
84: from sklearn.metrics import average_precision_curve
85: from sklearn.metrics import average_precision_curve
86: from sklearn.metrics import average_precision_curve
87: from sklearn.metrics import average_precision_curve
88: from sklearn.metrics import average_precision_curve
89: from sklearn.metrics import average_precision_curve
90: from sklearn.metrics import average_precision_curve
91: from sklearn.metrics import average_precision_curve
92: from sklearn.metrics import average_precision_curve
93: from sklearn.metrics import average_precision_curve
94: from sklearn.metrics import average_precision_curve
95: from sklearn.metrics import average_precision_curve
96: from sklearn.metrics import average_precision_curve
97: from sklearn.metrics import average_precision_curve
98: from sklearn.metrics import average_precision_curve
99: from sklearn.metrics import average_precision_curve
100: from sklearn.metrics import average_precision_curve

```

Tweaked the jail config to monitor MySQL login attempts specifically. This included adjusting the ban time and the number of allowed failures.

```

# Set the mail server globally (can be overridden in the server's mail section from: ps
# corresponding jail-id my.jail.id):
action:mail_list_m = black list de (email("kismet"), services("kismet..."), address("black list de, myjail"), agent("TotalScanAgent"))
# Hosts list via sshutils.cas.
# See action:sshutils.cas.conf for usage example and details.
action:sshutils_cas = sshutils

# Change default action. To change, but override value of 'action' with the
# corresponding to the new chosen action (default is 'action_m, action_m, src is jail.id,
# action is the default) or per specific section
action = Section[""]

# Jails
#
# SSH servers
#
[conf]
enable = true
password = foo
portnumber = 22
ipaddress = 127.0.0.1

```

Confirmed that the jail was enabled and running. The MySQL filter was active and ready to take action if someone tried to mess around.

[illegible]

Installed Auditd so I could track important system events, especially related to MySQL file access and sudo commands.

```

# Create a connection to the database
conn = psycopg2.connect(
    dbname='postgres',
    user='postgres',
    password='postgres',
    host='localhost',
    port='5432')

# Create a cursor object
cursor = conn.cursor()

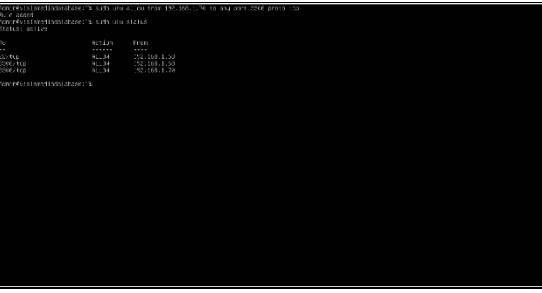
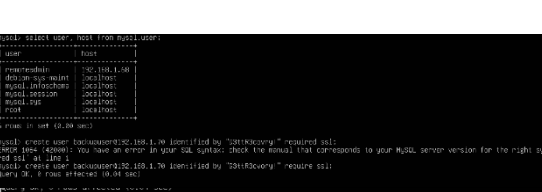
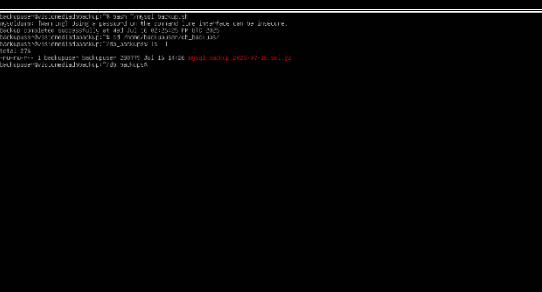
# Create a table
cursor.execute('CREATE TABLE IF NOT EXISTS users (id SERIAL PRIMARY KEY, name VARCHAR(50), email VARCHAR(100))')

# Insert data
cursor.execute('INSERT INTO users (name, email) VALUES (%s, %s)', ('John Doe', 'john.doe@example.com'))

# Commit the transaction
conn.commit()

# Close the cursor and connection
cursor.close()
conn.close()

```

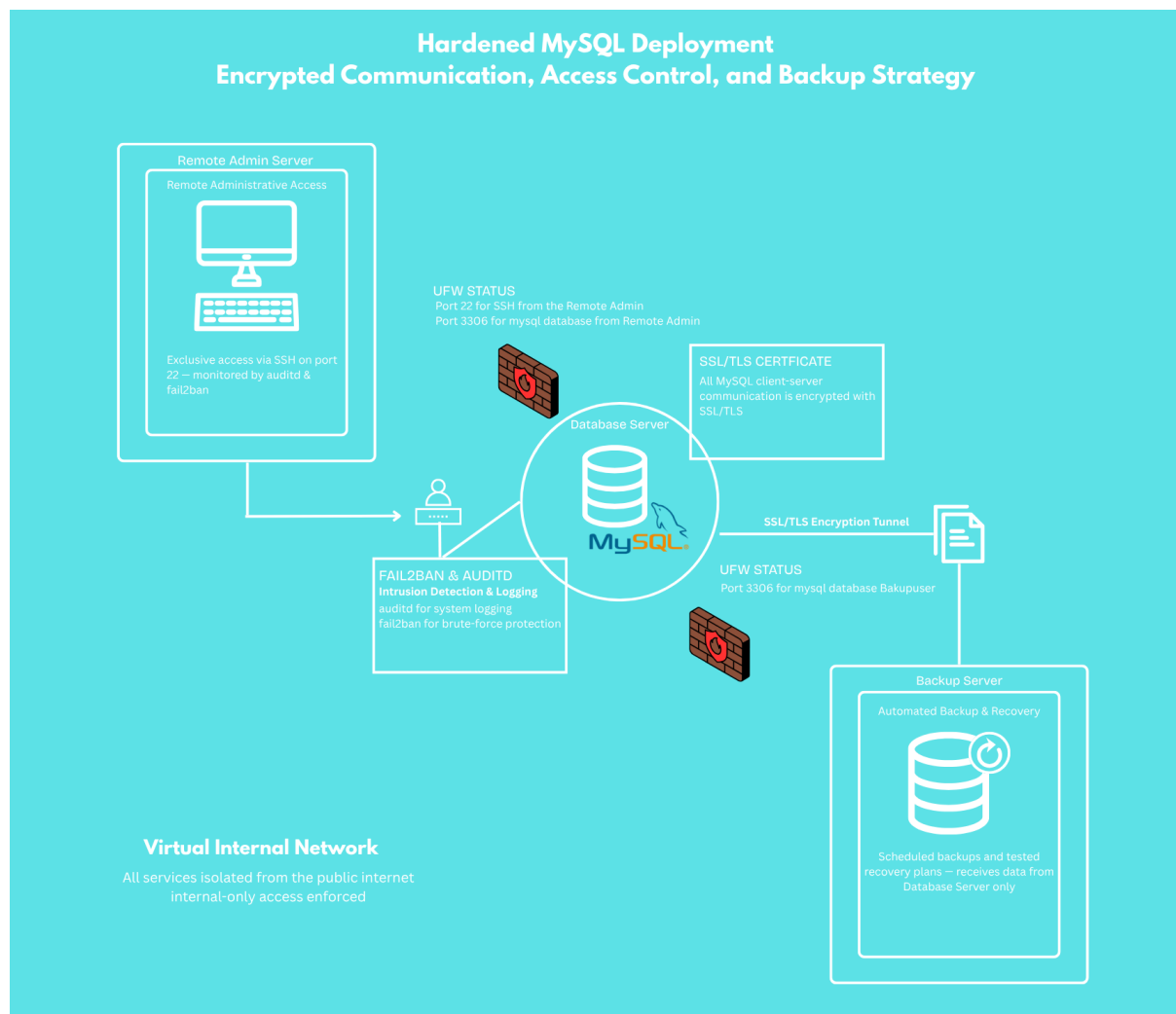

<p>Added Backup Server to the UFW</p> <p>On the MySQL server, I updated the firewall to only allow MySQL connections from the backup server's IP address.</p>	
<p>Create a MySQL user for Backup Server</p> <p>Created a new MySQL user specifically for backups. This user only works from the backup server's IP and is forced to use SSL to connect.</p>	
<p>Granting Backup Server Privileges</p> <p>Gave the backup user just enough privileges to read and export data — nothing more. It can't modify anything, just dump the database.</p>	
<p>Backup Server Connecting to MySQL</p> <p>Tested the connection using the client-side certificates. Confirmed that the backup server could only connect if SSL was used, which was exactly what I wanted.</p>	
<p>Create mysqldump script</p> <p>Wrote a bash script on the backup server that uses mysqldump to export the full database and save it with a timestamp. This makes it easy to keep track of backup versions.</p>	
<p>Test Script</p> <p>Ran the script manually to make sure it works. The output was clean, the file was generated, and everything looked good.</p>	
<p>Set Crontab -e to automate backup</p> <p>Added the script to the system's crontab so it runs automatically at 2:30 PM every day. Set it and forget it.</p>	

Crontab -l to verify it has been set
Listed the current cron jobs to confirm the
backup task was properly registered.

```
crontab -l
# Sample crontab for backup task
0 2 * * * /usr/bin/rsync -e 'ssh -C -o "ProxyCommand ssh -W %h:%p"' -u backupuser backupserver:/data/mysql:/data/mysql --exclude='*.log' /data/mysql --delete --recursive /data/mysql
# Backup task will be running based on the user's system
# Backup of the database data (including backups) is sent through
# email - to user the cron job it's defined in the file /etc/crontab
# For more information see: https://www.debian.org/doc/manuals/debian-faq/chapter8.en.html
# To P. see the cron command
0 2 * * * /usr/bin/rsync -e 'ssh -C -o "ProxyCommand ssh -W %h:%p"' -u backupuser backupserver:/data/mysql:/data/mysql --exclude='*.log' /data/mysql --delete --recursive /data/mysql
# Backup task will be running based on the user's system
```

Project Overview

This deployment was all about locking down a MySQL environment inside a private virtual network. I used a Remote Admin Server to handle all database management, with tight firewall rules allowing only necessary ports. MySQL connections were secured using SSL/TLS encryption, and system activity was monitored with Fail2Ban and Auditd to catch intrusions and log sensitive actions. A second server handled automated, encrypted backups, pulling data securely through an SSL tunnel. Everything stays off the public internet — internal-only communication is enforced end to end.



Conclusion

The MySQL infrastructure I built is already secure and isolated, but there's always room to push things further. Here are five key improvements I'd consider adding next, all focused specifically on tightening security:

Enable Multi-Factor Authentication (MFA) for SSH Access

Adding 2FA to the Remote Admin Server's SSH login would make brute-force or credential-based attacks much harder to pull off.

Implement Centralized Log Aggregation & Alerting

By collecting logs from all servers into a single dashboard, it becomes easier to detect threats in real time and respond faster.

Off-Site Encrypted Backup Storage

Right now, backups are stored locally. Pushing encrypted copies to a remote location ensures recovery is possible even if the local server is compromised or damaged.

Encrypt MySQL Data at Rest

Encrypting the storage volumes or data directories protects sensitive information even if someone gets direct disk access.

Set Up Certificate Expiry Monitoring

Keeping an eye on SSL certificate expiry dates helps avoid downtime or accidental unencrypted connections when certs silently expire.