MySQL Backup and Security

Best practices on how to run MySQL on Linux in a secure way

Lenz Grimmer <lenz@mysql.com>

Free and Open Source Software Conference
Bonn/Rhein-Sieg, Germany

24./25. June 2006

MySQL AB

Introduction

In this session you will learn best practises on how to configure and run MySQL on Linux in a secure way. It will give an overview about the security mechanisms built into MySQL and how they can be improved and accompanied by security mechanisms provided by the Linux Operating system.

In addition to to improving the security of a MySQL installation, several MySQL backup possibilities/tools and strategies for Linux are discussed.

Session content

- Improving MySQL Security
  - On the MySQL server level
  - One the Linux OS level
- MySQL backup methods
  - Physical vs. logical backup
  - OSS tools suitable for backup purposes
  - Commercial backup solutions

Improving MySQL security

- Securing MySQL is an essential part of the post-installation process
- While the default installation is pretty secure by itself already, some additional steps have to be performed
- In addition to the facilities provided by MySQL itself, make use of additional security features provided by the OS

MySQL Server post-installation

- Make sure to set a password for the **root** account

```
$ mysql -u root mysql
mysql> SET PASSWORD FOR
  root@localhost=PASSWORD('new_password');
```

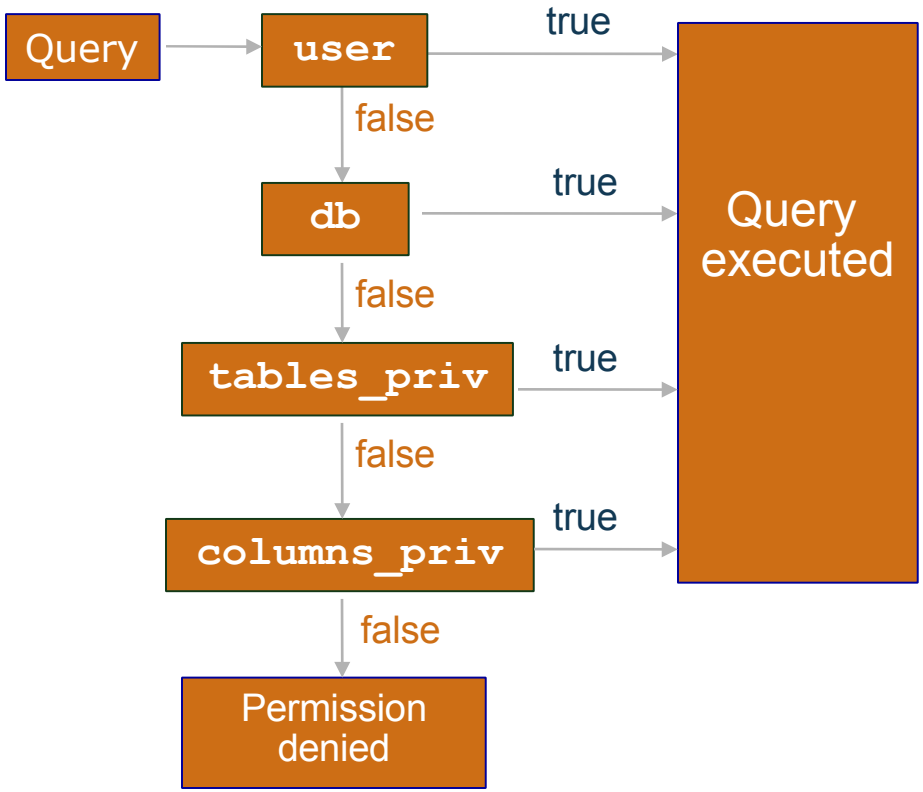
- Remove the **anonymous** account or assign a password to it
- Remove the **test** database, if you don't need it
- All the above steps can be performed by running the **mysql_secure_installation** script included in the Unix distributions

Access Control Check

- **Connect**
  - When a user connects the server checks in the **`user`** table to see if it can find a matching entry for the **username**, **host** and **password**
- **Query**
  - When a query is executed the server checks the **`user`**, **`db`**, **`tables_priv`** and **`column_privs`** tables

Query Access Control

Do you have sufficient privileges to execute the query?



MySQL Server security hints

- Use the **`bind-address`** option in **`my.cfg`** to bind the TCP port to a specific interface (e.g. **`127.0.0.1`**)
- Consider the **`skip-networking`** option, which only allows connections via the local socket file
- Allow access from selected hosts only
- Restrict access to the **`mysql.user`** table to the **`root`** user
- Learn how to use the **`SHOW GRANTS`**, **`SET PASSWORD`** and **`GRANT/REVOKE`** statements
- Use phpMyAdmin or MySQL Administrator for the user administration

MySQL Server security hints

- Restrict **`PROCESS/SUPER/FILE`** privileges to a minimum
- Do not store any plain-text passwords in your database. Instead, use **`MD5()`**, **`SHA1()`** or some other one-way hashing function.
- Disable **`LOAD DATA LOCAL`** by setting **`local-infile=0`** in **`my.cnf`**
- Always use a non-privileged account to run **`mysqld`**

MySQL Server security hints

- For the paranoid:
  - replace the root account with a different, harder to guess one to avoid brute-force dictionary attacks
  - make sure to remove or clean the history file of the mysql command line client (**`~/.mysql_history`**), if you used it to edit or add user accounts/passwords

Views and Stored Procedures

- **Views** can be used to restrict access to certain columns of tables
- **Stored Procedures** can be used to shield the tables from being accessed/modified by the user/application directly
- Plug: Now available in MySQL 5.0!

Improving access restrictions

- Lock down the permissions on the data directory with **`chown`** and **`chmod`**
  - users won't **corrupt** table data
  - users won't **see data** they aren't supposed to see
- The log files must also be kept secure:
  - users might again **see data** they aren't supposed to see
  - queries such as **`GRANT`** are stored in the logfiles, anyone with log file access could then **obtain user passwords**
- Generally don't allow regular user shell logins to the DB server

Reducing security risks with Linux

- Use iptables to firewall the server
- Run MySQL in a `chroot()` jail
- Enable SELinux or Novell AppArmor
- Run the MySQL server in a virtual machine
  - Xen
  - UML (User Mode Linux)
  - VMware / Parallels

Securing data and communication

- Encrypt network traffic
  - OpenSSL
  - SSH tunnel
  - OpenVPN
  - Cipe
- Encrypt the Data Directory
  - cryptoloop devices
  - dm_crypt kernel module

Backing up MySQL data

- When do you need backups?
- What needs to be backed up?
- When should backups be performed?
- Where will the backups be stored?
- How can backups be performed?

When Do You Need Backups?

- Hardware failure
  - When a system crash occurs some of the data in the databases may be lost
  - A hard-disk failure will most certainly lead to lost data
- User failure
  - A user may issue DROP TABLE or DELETE FROM statements that he or she later regrets
  - Someone (an administrator?) might try to edit the table files with text editors, usually leading to corrupt tables

What needs to be backed up?

- Database content
  - for full backups
  - logical or physical backup
- Log files
  - for incremental backups
  - point-in-time recovery

When should backups be performed?

- On a regular basis
- Not during high usage peaks (off hours)
- Static data can be backed up less frequently

Where to store backups?

- On the database server
  - At least on a separate file system/volume or hard disk drive
- Copied to another server
  - On or off site
- Backed up to tape/disk
  - Stored on or off site
- Choose multiple locations

The Data Directory

- By default all databases as well as most log and status files are stored in the data directory
- A default data directory is compiled into the server
  - **`/usr/local/mysql/data/`** (tarball installation)
  - **`/var/lib/mysql`** (RPM packages)
- The data directory location can be specified during server startup with **`--datadir=/your/path/`**
- If you don't know the location of the data directory you can find it out with:
  - **`mysql> SHOW VARIABLES like 'data%';`**

The Binary Log

- Contains all SQL commands that actually change data
- Also contains additional information on each query
  e.g. query execution time
- The binary log is not stored in text format, it is stored in a more efficient binary format
- You must use **`mysqlbinlog`** to access the log contents
- Turned on with **`--log-bin[=file_name]`**
- The update logs are created in sequence
  e.g. **`file_name-bin.001`**, **`file_name-bin.002`**, etc.
- The binary log is compatible with transactions
- mysqld creates a binary log index file which contains the names of the binary log files used

Managing The Binary Log

- The purpose of the Binary Log is to
  - Ease crash recovery
  - Enable replication
- **`SHOW MASTER LOGS`** shows all binary log files residing on the server
- With **`FLUSH LOGS`** or when restarting the server a new file is used
- **`RESET MASTER`** deletes all binary log files
- **`PURGE MASTER`** deletes all binary log files up to a certain point

The Error Log

- When the server is started with **`mysqld_safe`** all the **error messages** are directed to the error log
- The log contains info on when **`mysqld`** was **started** and **stopped** as well as **errors** found when running

```
$ cat /var/log/mysql.err
000929 15:29:45  mysqld started
/usr/sbin/mysqld: ready for connections
000929 15:31:15  Aborted connection 1 to db: 'unconnected'
user: 'root' host: `localhost' (Got an error writing communication
    packets)
000929 15:31:15  /usr/local/mysql/bin/mysqld: Normal shutdown

000929 15:31:15  /usr/local/mysql/bin/mysqld: Shutdown Complete

000929 15:31:54  mysqld started
/usr/sbin/mysqld: ready for connections
```

mysqldump

- **mysqldump** dumps the table structure and data into SQL statements, which can be saved in files
  - **$ mysqldump mydb > mydb.20050925.sql**
- You can dump **individual tables** or **whole databases**
- The default output from **mysqldump** consists of **SQL statements**, **CREATE TABLE** statements for table structure and **INSERT** statements for the data
- **mysqldump** can also be used directly as input into another **mysqld** server (without creating any files)
  - **$ mysqldump --opt world | mysql -hwork.mysql.com world**

Recovering With Backups

Recovered database = Backup files + binary log

- In order to restore the tables to the state before a crash you will need both your **backup files** and the **binary log**
  - From the backup files you can restore the tables to the state they were at the **time of the backup**
  - From your **synchronised** binary logs you can extract the queries issued **between** the **backup** and **now**
- Beware, if you are recovering data lost due to **unwise** queries remember **not** to issue them again

Example SQL level restore

- Restore the last full backup

```
mysql < backup.sql
```

- apply all incremental changes done after the last full backup

```
mysqlbinlog hostname-bin.000001 |
mysql
```

MySQL table files backup

- Also called “physical” backup
- Database files can be simply be copied after issuing `FLUSH TABLES WITH READ LOCK;`
- The `mysqlhotcopy` Perl script automates this process (MyISAM table files only)
- Locking all tables for consistency can be expensive, if the file backup operation takes a long time

mysqlhotcopy

- **mysqlhotcopy** is a Perl script with which you can easily backup databases
- It can **only** be run on the **same** machine as where the databases are
- It does the following
  - **LOCK TABLES**
  - **FLUSH TABLES**
  - Copies the table files to the desired location with **cp** or **scp**
  - **UNLOCK TABLES**
- The user has to have **write access** to the target directory

Backing Up InnoDB Databases

- You can use the **`mysqldump --single transaction`** tool to make an on-line backup
- To take a '**binary**' backup, do the following:
  1. Shutdown the MySQL server
  2. Copy your **data** files, InnoDB **log** files, **`.frm`** files and **`my.cnf`** file(s) to a safe location
  3. Restart the server
- It is a **good** idea to backup with **`mysqldump`** also, since an error might occur in a binary file **without** you noticing it

OSS backup tools

- The usual suspects: **`cp`**, **`tar`**, **`cpio`**, **`gzip`**, **`zip`** called in a shell script via a **`cron`** job
- Use **`rsync`** or **`unison`** for bandwidth-friendly remote backups
- Complete network-based backup solutions like afbackup, Amanda or Bacula provide more sophisticated features (e.g. catalogs)

Linux backup support

- LVM snapshots
- DRBD (“RAID1 over the network”)
- Distributed file systems
  - OpenAFS
  - GFS
  - Lustre
  - Novell iFolder

Backup using LVM snapshots

- Linux LVM snapshots provide a very convenient and fast backup solution for backing up entire databases without disruption
- The snapshot volume does not need to be very large (10-15% are sufficient in a typical scenario)
- A backup of the files from the snapshot volume can be performed with any tool

Linux LVM snapshot creation

Basic principle:

```
mysql> FLUSH TABLES WITH READ LOCK
$ lvcreate -s --size=<size> --name=backup
<LV>
mysql> UNLOCK TABLES
$ mount /dev/<VG>/backup /mnt
$ tar czvf backup.tar.gz /mnt/*
$ umount /mnt
$ lvremove /dev/<VG>/backup
```

The mylvmbackup script

- A Perl script for quickly creating backups of MySQL server data files using LVM snapshots
- The LVM snapshot is mounted to a temporary directory and all data is backed up using the tar program
- Use of timestamped archive names allows you to run **`mylvmbackup`** many times without danger of rewriting old archives.
- requires Perl and the DBI and DBD::mysql modules
- Available from http://www.lenzg.org/mylvmbackup/

MySQL replication

- Backing up a replication slave is less time-critical (the master is not blocked for updates)
- A slave can use different storage engines
- One Master can replicate to many slaves
- Keep the limitations of MySQL replication in mind
- Make sure to back up the `master.info` and `relay-log.info` files as well as any `SQL_LOAD-*` files (if `LOAD DATA INFILE` is replicated)

Commercial backup solutions

- Acronis True Image
- ARCServe
- Arkeia
- InnoDB HotBackup
- SEP sesam
- Veritas vxfs snapshots

Backup Method Comparison

- The output from **`mysqldump`** is portable to any other DBMS (without the **`--opt`** option) whereas the copied files only work with MySQL
- The file copying methods are **much faster** than **`mysqldump`**
- So it comes down to **your** preferences:
  - Which **tool** do you prefer to use
  - Speed vs. portability

Backup Principles

- Perform backups regularly
- Turn on the binary update log
  - The update logs are needed to restore the database without losing any data
- Synchronise your update log files with your backup files
  - Use **`FLUSH LOGS`**
- Name your backups consistently and understandably
  - Include the date in the file name **`mydb.20050925.sql`**
- Store your backups on a different file system than where your databases are
- Backup your backup files with file system backups

General backup notes

- Putting the binary logs on a different file system (or even a different drive) than the data directory is recommended (increases performance and avoids data loss)
- Make sure the backup is consistent and complete!
- Define **backup schedules** and **policies** as well as **recovery procedures**
- **Test** that these actually work!

The MySQL Online Backup API

- Work is in in progress to define an API to perform a streaming MySQL online backup, independent of the Storage Engine
- Transactional tables will contain data only from committed transactions
- Non-transactional tables will contain data only from completed statements
- Referential integrity will be maintained between all tables backed up with a specific backup command
- The spec is now available for comments/review on MySQL Forge: http://forge.mysql.com/wiki/OnlineBackup

Thank you!

Questions, Comments?

Lenz Grimmer <lenz@mysql.com>