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

Lenz Grimmer <lenz.grimmer@sun.com>

DrupalCon 2008 Szeged, Hungary
28. August 2008

Sun Microsystems
Introduction

- Learning best practices about configuring and running MySQL in a secure way
- Security mechanisms built into MySQL and how to improve them by using OS features
- Discussion of MySQL backup possibilities/tools and strategies
  - Physical vs. logical backup
  - OSS tools suitable for backup purposes
  - Commercial backup solutions
Improving MySQL security

• Essential part of the post-installation process: Security
• Default installation pretty secure already
• Some additional steps have to be performed
• Also utilize additional security features provided by the OS (where applicable)
MySQL Server post-installation

- 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 (you usually don't need it)

- `mysql_secure_installation` script does all of the above (Unix only)
Access Control Check

• Connect
  > Server checks in the user table for a matching entry for the username, host and password

• Query
  > Server checks the user, db, tables_priv and column_prives tables
Query Access Control

Do you have sufficient privileges to execute the query?

- Query
  - user: true
  - db: true
  - tables_priv: true
  - columns_priv: true

Permission denied
MySQL Server security hints

- **bind-address** option in `my.cnf` binds the TCP port to a specific interface (e.g. `127.0.0.1`)
- **skip-networking** option 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 user administration
- Do **not** edit `mysql.user` directly!
MySQL Server security hints

- Restrict `PROCESS/SUPER/FILE` privileges to a minimum.
- Do not store plain-text passwords in the database. Use `MD5()`, `SHA1()` or some other one-way hashing function instead.
- Disable `LOAD DATA LOCAL` by setting `local-infile=0` in `my.cnf`.
- Always run `mysqld` using a non-privileged user account.
MySQL Server security hints

• For the paranoid:
  > replace the **root** account with a different (harder to guess) one to avoid brute-force dictionary attacks
  > remove/clean the client's history file (~/.mysql_history), if you edited or added user accounts/passwords
Views and Stored Procedures

- **Views** to restrict access to certain columns of tables
- **Stored Procedures** shield tables from being accessed/modified by the user/application directly
- Available since MySQL 5.0.x
Improving access restrictions

• Lock down permissions on the data directory with `chown` and `chmod`
  > users can’t **corrupt** table data
  > users can't **access data** they aren’t supposed to see

• 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
d  file access could then **obtain user passwords**

• Generally don't allow shell logins to the DB server for normal
  users
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 / Sun xVM
  > Solaris Zones/Container
  > UML (User Mode Linux)
  > VMware / Parallels / VirtualBox
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
  > A system crash may cause some of the data in the databases to be lost
  > A hard-disk failure will most certainly lead to lost data

• User/Application failure
  > Accidental DROP TABLE or malformed DELETE FROM statements
  > Editing 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

• Databases and most log and status files are stored in the data directory by default

• Default directory compiled into the server
  > /usr/local/mysql/data/ (tarball installation)
  > /var/lib/mysql (RPM packages)

• Data directory location can be specified during server startup with
  --datadir=/path/to/datadir/

• Find out the location by asking the server
  mysql> SHOW VARIABLES like 'data%';
The Binary Log

- Contains all SQL commands that change data
- Also contains additional information about each query (e.g. query execution time)
- Binary log is stored in an efficient binary format
- Use `mysqlbinlog` to decipher the log contents
- Log turned on with `--log-bin[=file_name]`
- Update logs are created in sequence e.g. `file_name-bin.001`, `file_name-bin.002`, etc.
- Binary log is transaction-compatible
- `mysqld` creates binary log index file which contains names of the binary log files used
Managing The Binary Log

- Purpose of the Binary Log:
  - Enable replication
  - Ease crash recovery

- **SHOW MASTER LOGS** shows all binary log files residing on the server

- **FLUSH LOGS** or restarting the server creates a new file

- **RESET MASTER** deletes all binary log files

- **PURGE MASTER** deletes all binary log files up to a certain point

- Don't delete logs that slaves still need
The Error Log

• When started with `mysqld_safe`, all 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

```bash
$ 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 table structure and data into SQL statements
  
  ```
  $ 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
  - **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 -h work.mysql.com world
  ```
Recovering With Backups

DB Recovery = Last full backup & binlog

• To restore tables to the state before a crash requires both the backup files and the binary log
  > Restore the tables to the state they were at the time of the backup from the backup files
  > Extract the queries issued between the backup and now from synchronised binary logs

• 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
- MyISAM Database files can simply be copied after issuing `FLUSH TABLES WITH READ LOCK;`
- The `mysqlhotcopy` Perl script automates this process
- 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

- Use `mysqldump --single transaction` 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
• `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
- Snapshot volume size does not need to be very large (10-15% are sufficient in a typical scenario)
- Backup of files from a snapshot volume can be performed with any tool
- I/O performance may be degraded due to the additional LVM logging
Linux LVM snapshot creation

Basic principle:

```bash
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 mylvmbbackup script

• A Perl script for quickly creating MySQL backups using LVM snapshots
• Snapshots are mounted to a temporary directory and all data is backed up using `tar` or `rsync`
• Timestamped archive names allow running `mylvmbbackup` many times without risking to overwrite old archives.
• Requires Perl, DBI and DBD::mysql
• Available from [http://www.lenzg.org/mylvmbbackup/](http://www.lenzg.org/mylvmbbackup/)
MySQL replication

• Backing up a replication slave is less time-critical (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
- Zmanda Recovery Manager (ZRM)
Backup Method Comparison

• Output from `mysqldump` is portable to any other DBMS (without the `--opt` option) whereas copied files only work with MySQL
• Full backups are expensive
• Restoring from logs can be tricky
• 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
  > Update logs are needed to restore the database without losing any data
- Synchronise update logs with the 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
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)
- Verify 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

• 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
• Now available on MySQL Forge: 
  http://forge.mysql.com/wiki/OnlineBackup
Thank you!

Questions, Comments?
Lenz Grimmer <lenz.grimmer@sun.com>