

# Securing MySQL: step-by-step

Artur Maj 2003-08-28

## 1. Introduction

MySQL is one of the most popular databases on the Internet and it is often used in conjunction with PHP. Besides its undoubted advantages such as easy of use and relatively high performance, MySQL offers simple but very effective security mechanisms. Unfortunately, the default installation of MySQL, and in particular the empty root password and the potential vulnerability to buffer overflow attacks, makes the database an easy target for attacks.

This article describes the basic steps which should be performed in order to secure a MySQL database against both local and remote attacks. This is the third and last of the [series](#) of articles devoted to securing Apache, PHP and MySQL.

### 1.1 Functionality

The article assumes that the Apache web server with the PHP module is installed in accordance with the previous articles, and is placed in the `/chroot/httpd` directory.

Apart from the above we assume the following:

- The MySQL database will be used only by PHP applications, installed on the same host;
- The default administrative tools, such as *mysqladmin*, *mysql*, *mysqldump* etc. will be used to manage the database;
- Remote data backup will be performed by utilizing the SSH protocol.

### 1.2 Security requirements

In order to achieve the highest possible level of security, the installation and configuration of MySQL should be performed in accordance with the following security requirements:

- MySQL database must be executed in a chrooted environment;
- MySQL processes must run under a unique UID/GID that is not used by any other system process;
- Only local access to MySQL will be allowed;
- MySQL root's account must be protected by a hard to guess password;

- The administrator's account will be renamed;
- Anonymous access to the database (by using the *nobody* account) must be disabled;
- All sample databases and tables must be removed.

## 2. Installing MySQL

Before we start securing MySQL, we must install the software on the server. As in the previous articles, we will start installation by creating a unique, regular group and user account on the operating system, which will be dedicated to the MySQL database:

```
pw groupadd mysql
pw useradd mysql -c "MySQL Server" -d /dev/null -g mysql -s /sbin/nologin
```

### 2.1 Compiling MySQL

We will compile and install MySQL software in the */usr/local/mysql* directory:

```
./configure --prefix=/usr/local/mysql --with-mysqld-user=mysql --with-unix-
socket-path=/tmp/mysql.sock --with-mysqld-ldflags=-all-static
make
su
make install
strip /usr/local/mysql/libexec/mysqld
scripts/mysql_install_db
chown -R root /usr/local/mysql
chown -R mysql /usr/local/mysql/var
chgrp -R mysql /usr/local/mysql
```

In general, the process of installing the server is almost identical to the one described in the MySQL manual. The only change is the use of a few additional parameters, specified in the *./configure* line. The most important difference is the use of *--with-mysqld-ldflags=-all-static* parameter, which causes the MySQL server to be linked statically. This will significantly simplify the process of chrooting the server, as described in Section 3. With regard to the other parameters, they instruct the *make* program to install the software in the */usr/local/mysql* directory, run the MySQL daemon with the privileges of the *mysql* account, and create the *mysql.sock* socket in the */tmp* directory.

## 2.2 Copy configuration file

After executing the above commands, we must copy the default configuration file in accordance with the expected size of the database (small, medium, large, huge). For example:

```
cp support-files/my-medium.cnf /etc/my.cnf
chown root:sys /etc/my.cnf
chmod 644 /etc/my.cnf
```

## 2.3 Start the server

At this point MySQL is fully installed and ready to run. We can start the MySQL server by executing the following command:

```
/usr/local/mysql/bin/mysqld_safe &
```

## 2.4 Test the connection

Try to establish a connection with the database as follows:

```
/usr/local/mysql/bin/mysql -u root mysql
```

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2 to server version: 4.0.13-log
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
```

```
mysql> show databases;
```

```
+-----+
```

```
| Database |
```

```
+-----+
```

```
| mysql   |
```

```
| test    |
```

```
+-----+
```

```
2 rows in set (0.00 sec)
```

```
mysql> quit;
```

Once the connection is successfully established, we can shutdown the database:

```
/usr/local/mysql/bin/mysqladmin -u root shutdown
```

and start securing the software. Otherwise, we should analyze the information stored in the `/usr/local/mysql/var/`hostname`.err` log file, and eliminate the cause of any problems.

### 3. Chrooting the server

The first step of securing MySQL is to prepare the chrooted environment, in which the MySQL server will run. The chrooting technique was described in detail in the first article of this series ("[Securing Apache: Step-by-Step](#)"), so if you are not familiar with the technique or why chrooting is recommended, please refer to that article.

#### 3.1 Operating system

Like in the previous articles, the target operating system is FreeBSD 4.7. However, the methods presented should also apply on most modern UNIX and UNIX-like systems.

#### 3.2 Prepare chroot environment

In order to prepare the chrooted environment, we must create the following directory structure:

```
mkdir -p /chroot/mysql/dev
mkdir -p /chroot/mysql/etc
mkdir -p /chroot/mysql/tmp
mkdir -p /chroot/mysql/var/tmp
mkdir -p /chroot/mysql/usr/local/mysql/libexec
mkdir -p /chroot/mysql/usr/local/mysql/share/mysql/english
```

#### 3.3 Set access rights

The access rights to the above directories should be set as follows:

```
chown -R root:sys /chroot/mysql
chmod -R 755 /chroot/mysql
chmod 1777 /chroot/mysql/tmp
```

### 3.4 Create directory structure

Next, the following files have to be copied into the new directory structure:

```
cp /usr/local/mysql/libexec/mysqld /chroot/mysql/usr/local/mysql/libexec/
cp /usr/local/mysql/share/mysql/english/errmsg.sys /chroot/mysql/usr/local/mysql/
share/mysql/english/
cp /etc/hosts /chroot/mysql/etc/
cp /etc/host.conf /chroot/mysql/etc/
cp /etc/resolv.conf /chroot/mysql/etc/
cp /etc/group /chroot/mysql/etc/
cp /etc/master.passwd /chroot/mysql/etc/passwords
cp /etc/my.cnf /chroot/mysql/etc/
```

### 3.5 Tighten passwords and groups

From the files: */chroot/mysql/etc/passwords* and */chroot/mysql/etc/group* we must remove all the lines except the mysql account and group. Next, we have to build the password database as follows (this applies only to FreeBSD):

```
cd /chroot/mysql/etc
pwd_mkdb -d /chroot/mysql/etc passwords
rm -rf /chroot/mysql/etc/master.passwd
```

### 3.6 Special considerations

As in case of the Apache web server, we have to create a special device file */dev/null*:

```
ls -al /dev/null
crw-rw-rw- 1 root sys 2, 2 Jun 21 18:31 /dev/null
mknod /chroot/mysql/dev/null c 2 2
chown root:sys /chroot/mysql/dev/null
chmod 666 /chroot/mysql/dev/null
```

We must also copy the *mysql* database, which contains grant tables created during MySQL installation:

```
cp -R /usr/local/mysql/var/ /chroot/mysql/usr/local/mysql/var
chown -R mysql:mysql /chroot/mysql/usr/local/mysql/var
```

### 3.7 Localization

If any language other than English will be used, we should copy the proper charsets from the `/usr/local/mysql/share/mysql/charsets` directory as well.

### 3.8 Test the configuration

At this point MySQL is ready to run in the chrooted environment. We can test if it runs correctly by executing the following command:

```
chrootuid /chroot/mysql mysql /usr/local/mysql/libexec/mysqld &
```

If any error occurs, we should use the `truss` command or an alternative, such as `ktrace/kdump`, `strace`, etc. This will help us to determine and eliminate the cause of the problems.

Notice, that in order to run the `mysqld` process, the `chrootuid` program was used instead of `chroot`, as in case of Apache or PHP. The main difference is that `chrootuid` changes the owner of the executing process. In our example, `mysqld` is being executed in a chrooted environment, but the owner of the process is not `root`, but `mysql` user. The `chrootuid` is not installed by default in many operating systems and it may be necessary to download and install this program manually. The `chrootuid` software can be downloaded [here](#).

## 4. Configuring the server

The next step is to configure the database server in compliance with our security requirements.

In case of default installation of MySQL, the main configuration file is `/etc/my.cnf`. In our case, however, because of running the server in a chrooted environment, we will use two configuration files: `/chroot/mysql/etc/my.cnf` and `/etc/my.cnf`. The first one will be used by MySQL server, and the latter will be used by MySQL tools (e.g. `mysqladmin`, `mysql`, `mysqldump` etc.). In both cases, some configuration changes will be required.

### 4.1 Disable remote access

The first change applies to the 3306/tcp port, on which MySQL listens by default. Because,

according to the initial assumptions, the database will be used only by locally installed PHP applications, we can freely disable listening on that port. This will limit possibilities of attacking the MySQL database by direct TCP/IP connections from other hosts. Local communication will be still possible through the *mysql.sock* socket. In order to disable listening on the mentioned port, the following parameter should be added to the *[mysqld]* section of */chroot/mysql/etc/my.cnf*:

```
skip-networking
```

If, for some reason, remote access to the database is still required (e.g. to perform remote data backup), the SSH protocol can be used as follows:

```
backuphost$ ssh mysqlserver /usr/local/mysql/bin/mysqldump -A > backup
```

## 4.2 Improve local security

The next change is to disable the use of *LOAD DATA LOCAL INFILE* command, which will help to prevent against unauthorized reading from local files. This matters especially when new SQL Injection vulnerabilities in PHP applications are found.

For that purpose, the following parameter should be added in the *[mysqld]* section in */chroot/mysql/etc/my.cnf*:

```
set-variable=local-infile=0
```

In addition, to make the use of the database administrative tools convenient, the following parameter should be changed in the *[client]* section of */etc/my.cnf*:

```
socket = /chroot/mysql/tmp/mysql.sock
```

Thanks to that, there will be no need to supply the *mysql*, *mysqladmin*, *mysqldump* etc. commands with the *--socket=/chroot/mysql/tmp/mysql.sock* parameter every time we run these tools.

## 4.3 Change admin password

One of the most important steps in securing MySQL is changing the database administrator's

password, which is empty by default. In order to perform that, we should run MySQL (if it is not already running):

```
chrootuid /chroot/mysql mysql /usr/local/mysql/libexec/mysqld &
```

and change the administrator's password as follows:

```
/usr/local/mysql/bin/mysql -u root
mysql> SET PASSWORD FOR root@localhost=PASSWORD('new_password');
```

It is good practice not to change passwords from the command line, for example, by using the "*mysqladmin password*" command. This is especially important when other users work on the server. In that case the password could be easily revealed, e.g. by using the "*ps aux*" command or reviewing history files (*~/.history*, *~/.bash\_history* etc), when improper access rights are set to them.

## 4.4 Remove default users/db

Next, we must remove the sample database (test) and all accounts except the local root account:

```
mysql> drop database test;
mysql> use mysql;
mysql> delete from db;
mysql> delete from user where not (host="localhost" and user="root");
mysql> flush privileges;
```

This will prevent the database from establishing anonymous connections and -- irrespective of the *skip-networking* parameter in */chroot/mysql/etc/my.cnf* -- remote connections as well.

## 4.5 Change admin name

It is also recommended to change the default name of administrator's account (*root*), to a different, harder to guess one. Such a change will make it difficult to perform brute-force and dictionary attacks on the administrator's password. In this case the intruder will have to guess not only the password, but first and foremost, the name of the administrator's account.

```
mysql> update user set user="mydbadmin" where user="root";  
mysql> flush privileges;
```

## 4.6 Remove history

Finally, we should also remove the content of the MySQL history file (`~/.mysql_history`), in which all executed SQL commands are being stored (especially passwords, which are stored as plain text):

```
cat /dev/null > ~/.mysql_history
```

## 5. Communication between PHP and MySQL

In the previous article ("[Securing PHP: Step-by-Step](#)"), I mentioned about the communication problem between PHP and MySQL when one of these programs is being executed in a chrooted environment. Because locally PHP communicates with MySQL by using the `/tmp/mysql.sock` socket, placing PHP in the chrooted environment means that they cannot communicate with each other. To solve that problem, each time we run MySQL we must create hard link to the PHP chrooted environment:

```
ln /chroot/mysql/tmp/mysql.sock /chroot/httpd/tmp/
```

Note that the `/chroot/mysql/tmp/mysql.sock` socket and the `/chroot/httpd/tmp` directory must be physically placed on the same filesystem. Otherwise the programs will not be able to communicate with each other -- hard links do not work between filesystems.

## 6. Final steps

At this point we can create all databases and accounts which will be used by specific PHP applications. It should be emphasized that these accounts should have access rights only to the databases which are used by the PHP applications. In particular, they should not have any access rights to the `mysql` database, nor any system or administrative privileges (FILE, GRANT, ALTER, SHOW DATABASE, RELOAD, SHUTDOWN, PROCESS, SUPER etc.).

At last, we should also create a shell script that will be used to run MySQL during operating system start up. Sample script is shown below and also available for [download](#):

```
#!/bin/sh
```

```
CHROOT_MYSQL=/chroot/mysql
```

```
CHROOT_PHP=/chroot/httpd
```

```
SOCKET=/tmp/mysql.sock
```

```
MYSQLD=/usr/local/mysql/libexec/mysqld
```

```
PIDFILE=/usr/local/mysql/var/`hostname`.pid
```

```
CHROOTUID=/usr/local/sbin/chrootuid
```

```
echo -n " mysql"
```

```
case "$1" in
```

```
start)
```

```
    rm -rf ${CHROOT_PHP}/${SOCKET}
```

```
    nohup ${CHROOTUID} ${CHROOT_MYSQL} mysql ${MYSQLD} >/dev/null 2>&1 &
```

```
    sleep 5 && ln ${CHROOT_MYSQL}/${SOCKET} ${CHROOT_PHP}/${SOCKET}
```

```
    ;;
```

```
stop)
```

```
    kill `cat ${CHROOT_MYSQL}/${PIDFILE}`
```

```
    rm -rf ${CHROOT_MYSQL}/${SOCKET}
```

```
    ;;
```

```
*)
```

```
    echo ""
```

```
    echo "Usage: `basename $0` {start|stop}" >&2
```

```
    exit 64
```

```
    ;;
```

```
esac
```

```
exit 0
```

In case of our FreeBSD system, the above script should be placed in the */usr/local/etc/rc.d* directory, under the name of *mysql.sh*.

## 6.1 Summary

Applying the methods described in the article allows us to significantly increase the security of

MySQL. By running the database in a chrooted environment, disabling listening on 3306/tcp port and applying strong passwords to users' accounts we can make the database immune to a many of the attacks that would be possible with the default installation.

Although no method will let us achieve 100% security, applying the outlined methods will at least limit attack possibilities from users who visit our web servers with unfair intentions.

## Relevant Links

[Securing Apache: Step-by-Step](#)

[Securing PHP: Step-by-Step](#)

[MySQL Website](#)

[chrootuid package](#)

## About the author

[Artur Maj](#) works as a Principal Software Engineer for Oracle Corporation, in the EMEA Mobile, Wireless & Voice Center of Expertise. He is experienced in designing computer systems, performing security audits as well as providing security training. He is also author of many articles and publications devoted to securing computer systems and software against intruders.

View [more articles](#) by Artur Maj on SecurityFocus.

*Comments or reprint requests can be sent to the [editor](#).*

[Privacy Statement](#)

Copyright 2006, SecurityFocus