

# Secure MySQL Database Design

*Kristy Westphal* 2003-02-18

## Secure MySQL Database Design

by *Kristy Westphal*

last February 18, 2003

---

When it comes to installing software, secure design is often the last consideration. The first goal is usually just to get it to work. This is particularly true of databases. Databases are commonly referred to the keys to the kingdom: meaning that once they are compromised, all the valuable data that is stored there could fall into the hands of the attacker. With this in mind, this article will discuss various methods to secure databases, specifically one of the most popular freeware databases in use today, MySQL.

### Introduction to MySQL

MySQL can be found at <http://www.mysql.com> or at <http://sourceforge.net/projects/mysql/>.

MySQL is used in over 4 million installations all over the world. It is licensed under both GNU GPL (General Public License) and commercial licenses, depending upon what level of support that you require. It has a large user community, which makes it somewhat easier to use under the GPL license. There are actually four versions of MySQL:

- **MySQL Standard** includes the standard storage engine, as well as the InnoDB storage engine, which is touted as a "transaction-safe, ACID-compliant database" with some additional features over the standard version.
- **MySQL Pro** is the commercial version.
- **MySQL Max** includes the more technologically advanced features that are available during early access programs.
- **MySQL Classic** is the standard storage engine without the InnoDB engine. This is another commercial version.

Most of the recommendations discussed in this article apply to all versions of MySQL unless otherwise noted.

### Introducing Security Into MySQL Design

As with securing a network, securing a database by looking at the various layers that are

involved is an effective approach. Security of databases can be defined as preventing unauthorized or accidental disclosure, alteration, or destruction of data [2]. In addition, the confidentiality of data that exists in the database must be considered, as should the availability of that data. The following section will discuss a secure database design; while not all-inclusive, it should provide a good, basic starting point.

## Three-Tier Design

Also referred to as n-tier design, this design incorporates the three layers of a Web application running on different servers, usually set apart by firewalls that have specific rules to only let traffic through to the specific port on a specific server at whichever layer that the user is trying to access:

Internet -> Firewall -> Web -> Firewall -> Application -> Firewall -> Database

Something else that it should demonstrate is that it is very costly to implement such a design because firewalls and servers are not cheap. Oftentimes, a sys admin will choose a compromise, combining the application and database servers. This isn't ideal from a security perspective; nevertheless, it is a vast improvement over leaving a sensitive database facing the Internet directly. The point is that if one of the layers closest to the Internet is compromised, then several more layers still need to be compromised before access to the vital information can be gained.

## Access Control

Access to information contained in the tables must be properly regulated. This can be done with control over direct access to the tables, and also through views. Views and privileges assigned to the views can be created to limit users to only see specified portions of data contained within a table [2]. Through the use of the selects, projections and joins, existing relations between tables in a relational database, as well as a single table, can be created. Control over the `read`, `insert`, `update` and `delete` commands must also be assigned appropriately within those views.

## Roles

Role-based authentication should be considered when adding access to any database. Typical roles for access include administrator, user, programmer and operator. For the first three roles,

it is fairly obvious what access should be granted; it is the operator role that can be a sticking point. Operators are expected to play an essential part in the production operation of a system, yet they are often restricted in what type of access they are granted. Segregation of duties should be considered in the operator role, instead of just granting one operator control over an entire process. Operators' roles do need to be carefully defined and kept within the realm of production support as much as possible. Furthermore, all roles should have logging enabled to keep track of what occurs [3].

## **Integrity**

Another key ingredient in database design is data integrity, or ensuring that the data that is stored in the database is in fact valid and accurate. It is best to determine very early in the design process that it will be responsible for ensuring the integrity of the database. No matter the sensitivity of the data (credit card information vs. your record collection), if the data isn't right, then what good is the database? When the owner is determined, they should maintain this role and appropriate access only, not attempting to dole this out to others less it become diluted and possibly become corrupt.

A good process for ensuring the integrity of the data includes understanding what is processed and then identifying what can be considered personal, critical, or proprietary. As with any security issue, risk must be assigned according to the likelihood that something could occur to that data and the potential effect of such an occurrence. Most of all, accountability must be assigned and designed into the environment where the database resides. Otherwise, the goals of privacy and security cannot be met [3].

## **Encryption**

The sensitivity of the data will logically determine the need for the use of encryption. There are a few things to consider when thinking about implementing encryption:

1. Will the data stored in the database need to be encrypted or just the user passwords?
2. Will you need to encrypt the data only in the local instance of the database, or do you need to also encrypt the data in transit?

## **Change Control**

It is important to remember that changes made to the database, whether structural or to the

data itself, must be tracked and regulated by interested parties. Whether formal or informal, the process must be defined and followed by all roles defined in the database structure.

## Specific MySQL Security Considerations

Now that we have covered some of the general principals of database security, we can examine some specific considerations for the MySQL database. Please note that many variables that are mentioned in the following discussion are set in the "my.cnf" file. The location of this depends on how the MySQL database is installed. Essentially, you can create the file on your own, or use one of the handy sample files that come with the distribution (see the "support-files" directory). Then, if you would like the parameters to apply all MySQL users, you can place the "my.cnf" file in /etc. If you want the parameters to apply to specific users, then you can set the file in their respective home directory as ".my.cnf". Make sure that the appropriate permissions are applied to the file wherever it resides, ensuring that the unauthorized users cannot write to it.

A discussion of the basic post-installation configuration of MySQL is beyond the scope of this discussion. For that information, please refer to the MySQL documentation, [Post-Installation Set-up and Testing](#), and [Setting Up the Initial MySQL Privileges](#), as well as Ryan W. Maple's article [MySQL Security](#).

## The MySQL Permission Model

In order to fully implement a secure MySQL database, it is necessary to learn the MySQL access control system (your friends the GRANT and REVOKE commands). There are four privilege levels that apply:

1. Global: these privileges apply to all databases on a server.
2. Database: these privileges apply to all tables in a database.
3. Table: these apply to all columns within a table.
4. Column: these apply to individual columns in a table.

The usage of these commands is varied:

```
GRANT priv_type [(column_list)] [, priv_type [(column_list)] ...]
  ON {tbl_name | * | *.* | db_name.*}
  TO user_name [IDENTIFIED BY [PASSWORD] 'password']
  [, user_name [IDENTIFIED BY 'password'] ...]
```

```

[REQUIRE
  NONE |
  [{SSL| X509}]
  [CIPHER cipher [AND]]
  [ISSUER issuer [AND]]
  [SUBJECT subject]]
[WITH [GRANT OPTION | MAX_QUERIES_PER_HOUR # |
      MAX_UPDATES_PER_HOUR # |
      MAX_CONNECTIONS_PER_HOUR #]]]

```

```

REVOKE priv_type [(column_list)] [, priv_type [(column_list)] ...]
  ON {tbl_name | * | *.* | db_name.*}
  FROM user_name [, user_name ...]

```

The privileges can get very granular, so it is important that they are used in a well planned fashion. The types of privileges include:

- Alter
- Delete
- Create
- Drop
- Execute
- Select
- Update

Once a database is completely set up, these privileges should be reviewed prior to going to any usage of the database to ensure that the privileges were set up appropriately.

For instance, if you wanted to limit the alter privilege only to the user `kristyw` for table called `CreditCards`, you would use the command as follows:

```

Mysql> GRANT alter on CreditCards to kristyw
Mysql> IDENTIFIED by "password";

```

This could take some time if multiple privileges for the same user are to be added. In this case, wildcards can be used, but use caution in doing so! You never want to add more privilege than is necessary or intended. Further, if a user account is compromised, then the use of blanket permissions to numerous hosts can open up unexpected trust relationships between systems. Additionally, if the hostname is left blank for connections, which also effectively works as a

wildcard [7].

For example, say the user kristyw should now have all the privileges to everything in the database, as well as be required to connect to the database via an SSL connection:

```
Mysql> GRANT all on *.* to kristyw
Mysql> IDENTIFIED by 'password' REQUIRE SSL;
```

The wildcards that apply with the GRANT and REVOKE permissions include the "\*", which when used with grant privileges to \*.\* indicates global permissions, the "\_", which if not used with a "\" in front of it (as in "\\\_") could unintentionally indicate access to other databases, and lastly, the "%" can be used in hostnames.

Another privilege that can be assigned controls via GRANT and REVOKE is the PROCESS privilege, which should be restricted to only appropriate users. When used in the format: "mysqladmin processlist", disclosure of password information is possible. This is especially true if the user executed the query with the syntax of "UPDATE user SET password=PASSWORD ('not\_secure')" in their query. Furthermore, restrict the use of the FILE privilege. This privilege allows the assigned user to write a file wherever the mysqld daemon has privileges too. In addition, the FILE privilege can be used to view any file within the UNIX filesystem that the user has privileges to [7].

## More Advanced Tips...

If a database only needs to be accessed locally, TCP networking can be disabled. You can achieve this by editing the safe\_mysqld script (located in /mysql\_install\_dir/bin). Search for "skip-locking", and add the "skip-networking" flag to the beginning of the line that includes "-skip-locking":

```
--skip-networking --skip-locking > > $err_log 2> &1
```

```
--skip-networking --skip-locking "$@" > > $err_log 2> &1
```

Now no one will be able to remotely connect to the database [4].

Start up MySQL with the parameter to disable the use of symlinks (via the -skip-symlink option). This will prevent the possibility of escalated privileges given to the owner of whoever started the mysqld process. Ultimately this could result in accidental or deliberate overwriting of

files on your system, so it is best to just remove their usage.

To prevent a type of denial of service by one compromised or careless user account, you should restrict connections for a single user, by setting the `max_user_connections` variable in `mysqld`. These options can be viewed via the `SHOW VARIABLES` command, and can be updated via the `SET` command [7].

There are a few methods to encrypt stored data in a MySQL database: you can use the `ENCRYPT` or `ENCODE` commands. The difference between the two is that `ENCRYPT` uses the Unix `crypt` call, where as `ENCODE` uses a password provided in the command to encrypt the string. Both MD5 and SHA1 hash algorithms are available, as well as AES and DES. (Note: SHA1, DES and AES algorithms only available in version 4.0.2 and later).

By default, passwords are inserted into tables using encryption. Also by default, passwords in MySQL are unrelated to operating system passwords. There is no length limit on passwords in MySQL; they can be as short or as long as you want (however, the OS may restrict that length). User names can be up to sixteen characters, but can also be shorter. Therefore, any strict password parameters will have to be encouraged by setting policies and enforced by auditing. Overall, just make sure that all accounts do have passwords, just like you should on their operating system accounts.

## All the Other Goodies

Another thing to keep in mind when securing your database is all of the other possible tools that are installed on your server. If you are on a tight budget, and have placed your Web server with the database (which is still not recommended), then tools like Apache, PHP or Java may be loaded on the same server. If these tools are not kept up-to-date (just as with the OS), then possible exploits may apply, and the server may be vulnerable to intrusion.

The same principle applies to the other utility packages that may be loaded on your server, such as: SSH, `zlib`, or `wu-ftp`. Always remember to apply security checklists to your servers, know what is running on them, and keep up with the latest vulnerabilities.

We have considered many things here to design into the database itself, but one item that needs to be considered is how the traffic is transmitted between the client and the server. If the data is sensitive and/or going over the Internet, then SSL should be employed. Version 4.0 of

MySQL will satisfy this requirement. To have your version of MySQL use SSL, configure it with the following flags: `--with-vio --with-openssl`.

There are also ways to lock down the usage of SSL once you have it properly installed. If the `REQUIRE SSL` option is used, no non-SSL connections can be made to this server. Be cautious when employing the `REQUIRE X509` option, though, because its usage alone does not mean that the certificate will be validated, only that the user must have one. Other `REQUIRE` options must be set to have proper validation (e.g. `ISSUER`, `SUBJECT`). Lastly, `REQUIRE CIPHER` is the parameter that requires that certain ciphers and keylengths be used [6].

For older versions of MySQL, an encrypted SSH tunnel is a viable option.

## **And All That Other Stuff...**

Don't forget the other layers of your servers!!! When I say this, I mean that the security of the operating system, the server authentication, and the server access control must all be taken care of as well, because if these are weak, then why even bother securing your database?

## **Database Backups**

Another area that often gets lost in the layers of security is the critical area of database backup and recovery. As a part of whatever backup type is used, testing recovery of data is mandatory. Further, since version 3.23.47, checkpointing (where copies of the database are saved at defined times while processing) has been improved to be done more frequently, also easing the recovery process. With frequent checkpointing, as well as transaction logging (now available with InnoDB) and making regular backup copies, backup and recovery of databases is made more straightforward.

Specifically, the InnoDB transactional model allows for commit, rollback and crash recovery. By adding locking capabilities for users, having many users access the database at the same time becomes faster and more stable. To ensure that InnoDB is available with your installation, configure your package with the `'--with-innodb'` flags. You will also want to specify InnoDB options in your `'my.cnf'` file. Details on these set-up options can be found at the MySQL Documentation on [InnoDB Start-up Options](#).

## **Add-Ons**

As is the case with a lot of popular freeware tools, MySQL has spawned many other tools that can help improve the management of MySQL databases; thus, making the improving the security of the databases as well. If you are looking for a tool to help scan your network for blank MySQL passwords, try this [MySQL Network Scanner](#) script. It was originally compiled for Linux and to scan a class C network, but could be modified if needed.

There are several GUI consoles available to make the management of MySQL database easier. For instance, [MySQL Explorer](#) allows several management processes to be done via an interface that runs on several windows platforms. The MySQL team also has a version in beta called MySQL Control Center, and the source code is available here: [here](#). Just keep in mind when using these graphical tools and editors to help you manage a MySQL database that security needs to apply to them as well. This can be done through use of the ACLs to make sure that only certain servers can connect to your database on certain ports.

## Conclusion

Many of the standard secure database design principles apply to MySQL. Of course, it has many of its own intricacies that need to be understood and audited carefully before any database is fully implemented. Lastly, it is important to keep in mind that other layers of security apply when hosting a database, such as network and operating system security. The good news is that the makers of MySQL have an excellent documentation area on their [Web site](#) that, although sometimes cumbersome to navigate, is well stocked with information for the MySQL developer and administrator.

## References

[1] [Database Management and Design](#), Gary W. Hansen and James V. Hansen, Prentice Hall, 1992

[2] [A Primer on SQL](#), Roy Ageloff, Times Mirror/Mosby College Publishing, 1988

[3] [Database: Structure Techniques for Design, Performance and Management](#), 2nd Edition, Shaku Atre, John Wiley and Sons, 1988

[4] mysql security, Ryan W. Maple, <http://www.linuxsecurity.com/tips/tip-24.html>

[5] General Security Guidelines, [http://www.mysql.com/doc/en/General\\_security.html](http://www.mysql.com/doc/en/General_security.html)

[6] SSL Usage Requirements [http://www.mysql.com/doc/en/Secure\\_requirements.html](http://www.mysql.com/doc/en/Secure_requirements.html)

[7] How to Make MySQL Secure Against Crackers <http://www.mysql.com/doc/en/Security.html>

## Relevant Links

[MySQL Mailing Lists](#)

*MySQL*

[Privacy Statement](#)

Copyright 2006, SecurityFocus