Securing MySQL Servers

OSTconf Moscow

Marian Marinov <mm@yuhu.biz>

Who am I?

:)

Why would you need to secure your mysql installations?

Why?

- Passwords stored in plain text
- E-Mail addresses
- Physical addresses
- Phone numbers
- Transaction information
- Sessions
- etc.

Passwords ought to be hashed in the DB...

But this is not always the case :(

Hashing had to be implemented with bcrypt...

But they were implemented with MD5 without salt or all with the same salt.

Sensitive data should be symmetrically encrypted

Problems:

- emails ending with @tmp.com
- phones starting with +4911
- addr. starting with Pattern

Not so obvious locations for sensitive data:

- logs
- temporary data
- actual data directories (LOAD DATA)
- SELECT ... INTO OUTFILE

Now let's focus on the securing part of this talk :)

Sensitive data in logs

- binary.log - log_bin (bool)
  - log_bin_basename
- relay.log - relay_log (bool)
  - relay_log_basename
- general.log - general_log (bool)
  - general_log_file
- error.log - log_error (bool)
  - log_error_verbosity (3)
  - log_statements_unsafe_for_binlog

Sensitive data in logs

- slow.log - slow_query_log (bool)
  - slow_query_log_file
  - log_slow_admin_statements
  - log_slow_slave_statements
  - long_query_time

Sensitive data in temporary folders:

- tmpdir
  - temporary tables
  - any temp data
- innodb_tmpdir
  - temp sort files
- slave_load_tmpdir
  - replication
  - LOAD DATA

Sensitive data in the data directories – LOAD DATA/XML

local_infile

One can use the LOAD DATA/XML statement to actually read your binary data files into tables...

secure_file_priv

The FILE privilege...

Sensitive data in the data directories

- SELECT ... INTO OUTFILE

Create new files:

- init_file, local_infile and my.cnf
- secure_file_priv && FILE

Chrooting the MySQL daemon

```
chroot=/var/lib/mysql
```

Chrooting will:

- restrict FS access to the chroot dir
- prevent read/write to system files
- require SSL certs in the chroot dir
- restrict, where the temporary files can be created
- restrict the pid and log file locations

Firewalling the MySQL

- DO NOT PUT MySQL on unrestricted public interfaces

```
# iptables -N mysql
# iptables -A mysql -j ACCEPT -s IP_1
# iptables -A mysql -j ACCEPT -s NET_1
# iptables -A mysql -j DROP
# iptables -A INPUT -j mysql -p tcp --dport 3306
```

Firewalling the MySQL

- Only disallow specific user (**app_user**)

```
# iptables -A OUTPUT -j DROP -p tcp --dport 3306 -m
owner ! --uid-owner app_user
```

Firewalling the MySQL

- or more then one user, but not everyone:

```
# iptables -N mysql_out
# iptables -A mysql_out -j ACCEPT -m owner --uid-owner
app_user1
# iptables -A mysql_out -j ACCEPT -m owner --uid-owner
app_user2
# iptables -A mysql_out -j DROP
# iptables -I OUTPUT -j mysql_out -p tcp --dport 3306
```

Firewalling the MySQL

- or you want only one specific user, to be restricted from MySQL

```
# iptables -A OUTPUT -j DROP -p tcp --dport 3306 -m
owner --uid-owner dev_user1
```

Access to the MySQL socket

The effects of:

```
# chmod 600 /var/lib/mysql/mysql.sock
```

- Only root & mysql have access to it
- restrict all users
- use sudo for devs

```
dev_user1 ALL=(mysql) PASSWD:/usr/bin/mysql
```

The socket protection:

Your app needs access to the socket, so:

```
# groupadd web_app
# usermod -a -G web_app mysql
# usermod -a -G web_app app_user
# chmod 660 /var/lib/mysql/mysql.sock
# chgrp web_app /var/lib/mysql/mysql.sock
```

MySQL authentication

- never leave a user without a password
- try not use the % in the host part of an account
- hostnames instead of IPs for user authentication and set skip_name_resolve
- do not set old_passwords=0
  - (pre mysql 4.1, hashing func. was producing 16bytes hash string)

MySQL authentication

secure_auth is used to control if old_passwords=1(pre 4.1 hashing) can be used by clients

After 5.6.5, **secure_auth** is enabled by default

MySQL authentication

```
mysql> SELECT PASSWORD('mypass');
+--------------------+
| PASSWORD('mypass') |
+--------------------+
| 6f8c114b58f2ce9e   |
+--------------------+
```

- try to avoid the mysql_native_password plugin(which produces 41bytes hash string)

```
mysql> SELECT PASSWORD('mypass');
+-------------------------------------------+
| PASSWORD('mypass')                        |
+-------------------------------------------+
| *6C8989366EAF75BB670AD8EA7A7FC1176A95CEF4 |
+-------------------------------------------+
```

MySQL authentication

The mysql-unsha1 attack

y = SHA1(password)

On every connection the server sends a salt(s) and the client computes a session token(x)

x = y XOR SHA1(s + SHA1(y))

the server will verify it with:

SHA1(x XOR SHA1(s + SHA1(y))) = SHA1(y)

MySQL authentication

Now if you can sniff the salt(x) and have access to the SHA1(password), you don't need the password :)

MySQL authentication

Finally, the security of the hashed passwords...

On 12th of Jun this year, Percona published this blog post

- 8 chars cracked for 2h and less then 20$
- 8 chars for 2.8y if you use sha256 auth plugins

MySQL authentication

You may also want to consider moving your authentication out of MySQL.

For example on external LDAP server or using PAM.

Account security

- SUPER
- REPLICATION
  - CLIENT
  - SLAVE – can dump all of your data
- FILE – can read and write on the FS

Now let's go over some options, that I consider related to security

- chroot (we already covered that one)
  old_passwords & secure_auth (we already covered these)
  local_inifile & init_file
  plugin-dir
  skip-grant-tables
  skip_networking
  skip_show_database
  secure_file_priv
  safe-user-create
  allow-suspicious-udfs
  automatic_sp_privileges
  tmpdir = save_load_tmpdir
  default_tmp_storage_engine
  internal_tmp_disk_storage_engine

MySQL SQL Security

- SQL SECURITY
  - DEFINER vs. INVOKER

```
CREATE DEFINER = 'admin'@'localhost' PROCEDURE p1()
SQL SECURITY DEFINER
BEGIN
  UPDATE t1 SET counter = counter + 1;
END;
```

- Triggers and evnets are always executed with definer's context

Data encryption at rest

In 2016, both MariaDB and Percona published information on how to encrypt your DB.
This comes built-in in MySQL 5.7.

Disk encryption in MySQL is supported ONLY by InnoDB and XtraDB storage engines

Other limitations

- Galera gcache is not encrypted
- .frm files are not encrypted
- mysqlbinlogs can no longer read the files
- Percona XtraBackup can't backup encrypted data
- Audit plugin can't create encrypted output
- general and slow logs, can't be encrypted
- error.log is not encrypted

However I prefer using ecryptfs Or LUKS, so I can keep all the data and logs encrypted, not only the data inside the MySQL DataBases

Questions

Marian Marinov <mm@yuhu.biz>