MySQL Security Armoring Your Dolphin

Ernie Souhrada, Senior Consultant

Webinar Presentation

21 August 2013

Agenda

- Know Thy Presenter
- Security Is a State of Mind
- Hardening the Lower Layers
- Application Design Tips
- Don't Forget the Meatware
- Running MySQL Securely
- What's New in MySQL 5.6?

Know Thy Presenter

- Joined Percona in April 2012
- Mathematics / Political Science academic
- Coming up on 20 years in IT, 15 with MySQL
- 10-time DEFCON attendee
- Skier, Psytrancer, Technological Generalist
- Specialization is for insects

If Only It Were That Easy...

Security is a State of Mind: 1

- Jimmy Hoffa runs the only truly secure server in the world.

Security is a State of Mind: 2

- If “THEY” want your data badly enough, “THEY” will find a way to get it. Period.
- THEY could be anybody....
  - Hackers
  - Competitors
  - Disgruntled employees or ex-employees
  - The US National Security Agency (NSA)
    [although they probably have it already]

Security is a State of Mind: 3

- Most attackers are not the NSA.
- Start with risk analysis.
  - What's your data worth?
    - To you? To them? To someone else?
  - Consider worst-case scenarios, but also the actual likelihood of such an event.
  - What are you willing to do or give up to get where you need to be, security-wise?

Security is a State of Mind: 4

- Guiding Principles
  - Principle of Least Access
  - Deny by Default
  - Minimization of Attack Surface
  - No Easy Targets
  - Separation of Concerns
  - Protect the Audit Trail
  - Defense in Depth
  - Formalization

Hardening the Lower Layers: 1

- Physical Security
  - Multi-factor authentication
    - Biometrics
    - Card keys
    - Combinations
  - Maintain accurate visitor/access logs
  - Strict limits on personnel who have access
- Why? Physical access == GAME OVER
- Do you trust your infrastructure provider?

Hardening the Lower Layers: 2

- External firewalls > on-box iptables
  - Using both is **probably** overkill, but does guard against potential vulnerabilities in the HW device.
- Physically isolate your traffic
  - Separate VLANs is better than nothing
  - DB servers should NEVER have public IPs.
    - Ideally, they should have no public network access at all, but this is a management/ease-of-use tradeoff.
  - Egress firewalls can help mitigate the risk of data exfiltration.

Hardening the Lower Layers: 3

- Use in-flight encryption technologies
  - VPN (OpenVPN, Cisco, Juniper, etc.)

    http://openvpn.net/
  - SSH, SSH tunnels

    http://www.linuxjournal.com/content/ssh-tunneling-poor-techies-vpn
  - MySQL client connections over SSL
  - MySQL replication over SSL
  - Tor

    (If you're really paranoid and don't care about performance.)

JUST SAY “NO” TO PLAINTEXT!

- It's 2013, does anyone even still use telnet or rlogin anymore?! If so, they should be shot.
- There are encrypted alternatives for virtually every common unencrypted protocol out there. Use them.

- FTP → SFTP/SCP
- Telnet/Rlogin → SSH
- IMAP/POP → IMAPS/POPS
- NNTP → NNTPS
- SMTP → SMTPS
- HTTP → HTTPS

MySQL? YES! MySQL + SSL

Hardening the Lower Layers: 4

- Recommended OS for MySQL = Linux or some other Unix-like OS (No, Mac OS X does not count!)
- Avoid Microsoft Windows unless absolutely necessary.
- Why? Not necessarily for the reasons you might expect. Think about how well Windows fits the guiding principles by default. Then think about how well Linux/Unix does it.

Hardening the Lower Layers: 5

- Uninstall, don't just disable, unneeded services.
- Compilers are useful attack tools, remove those, too.
- Heavily restrict shell access. Normal user shell access is often enough for GAME OVER.
- The old Unix crypt() and MD5 hash functions are broken. Newer versions of Linux allow using much stronger password-hash functions, such as SHA512. Use them.

```
# authconfig --passalgo=sha512 --update
```

JUST SAY “YES!” TO SELinux!

- The bane of many an engineer's existence.
- Learn to make it work for you, don't just blindly turn it off.
- Yes, there is a performance hit (about 7-10%), but if you're pushing your system so hard that 10% matters, you have other problems.
- The RPM installation of MySQL Community and Percona Server both work fine with SELinux out of the box. However, if you don't like the default paths / ports....

HOWTO: Impose your will upon SELinux (for MySQL)

- It's mostly about the proper security context.
  - **mysqld_db_t:** MySQL data directories / files
  - **mysqld_var_run_t:** MySQL socket, PID file
  - **mysqld_log_t:** MySQL log files
  - **mysqld_etc_t:** MySQL configuration
  - **mysqld_exec_t:** MySQL daemon executables
  - **mysqld_safe_exec_t:** mysqld_safe wrappers
  - **mysqld_initrc_exec_t:** MySQL system initscript

- There are also a couple of SELinux booleans:
  - allow_user_mysql_connect (default = OFF)
  - mysql_connect_any (default = OFF)

- Plus SELinux managed ports:

```
# semanage port -l | grep -i mysql
mysqld_port_t                  tcp      1186, 3306, 63132-63164
mysqlmanagerd_port_t    tcp      2273
```

SELinux + MySQL: Examples

- Want to have MySQL listen on a different port? Easy.

  ```
  # semanage port -a -t mysqld_port_t -p XXXXX
  ```

- **HACK**: On CentOS 6.4, any port that's not listed in /etc/services can apparently be used without needing to run semanage.
- Want to move the MySQL data directory to /mnt/mysql ? No problem.

  ```
  # chcon -R -t mysqld_db_t  /mnt/mysql
  ```

- What about /home/mysql? Not so fast! The **chcon** approach doesn't work due to different restrictions on **/home** (home_root_t). If you want to use /home/mysql, you need to use semanage:

  ```
  # semanage fcontext -a -t mysqld_db_t "/home/mysql(/.*)?"
  ```

- The **semanage** approach is preferable to **chcon** anyway; although changes via **chcon** will persist upon reboot, they'll be overwritten if that part of the filesystem is ever re-labelled.

HOWTO: SELinux + MySQL: 3

- Complicated configuration with SELinux isn't easy or quick. Simple changes, however, are.
- But it can protect you against threats you don't even know exist or vulnerabilities that haven't been discovered yet.

  http://www.youtube.com/watch?v=bsa2yWfACbo&t=7m10s

- Just say YES! to SELinux!

Hardening the Lower Layers: 6

- Keep up-to-date with security advisories and patches to your OS and system software.
  - http://www.us-cert.gov/ncas
  - http://cve.mitre.org/
  - http://nvd.nist.gov/
- SSH key-based authentication is more secure than password-based authentication, but you have to protect your private keys.
- Allowing root to SSH directly into a machine is nice and convenient, but risky.

Application Design Tips: 1

- Fail securely.
  - An error in processing should be treated as if the operation was disallowed.
  - Don't give away information.
  - Examples:
    - A login page should **NOT** return both “user not found” and “password invalid”
    - If there's a failure authenticating the user due to a DB error, that should be treated equivalently to the user having entered incorrect information.

Application Design Tips: 2

- Don't connect to the DB as root!
  - Each application should have its own named user account with only the per-database privileges necessary to run.
  - The same goes for your sysadmins and DB users. Who's running that query that's been “Sending data” for the last 4 hours?

Application Design Tips: 3

- Don't store sensitive data in plaintext.
  - MySQL offers the AES_ENCRYPT() and AES_DECRYPT() functions, along with MD5() and SHA1() for hashing. Or, encrypt the data in your application.
- Don't send sensitive data in plaintext, either.
  - In 2013, there are still lots of websites that happily respond to a “forgot my password” request by sending your password to you in the clear.

Application Design Tips: 4

- Always validate and sanitize user input.
- This is important, so I'll say it again.
  - At best, garbage data in your database.
  - At worst, SQL injections, CSRF (Cross-Site Request Forgery), XSS attacks, and the like.
  - Example:
    - <script language=”javascript”>alert('oops!');</script>
  - Overly-simplistic filters can miss creative ways of encoding special characters.

Application Design Tips: 5

- Beware of Dynamic SQL.
  - This is dangerous:

    $query = “SELECT * FROM items WHERE owner = ' “

    . $userName . “ ' AND itemname = ' “ . $itemName + “ ' “;

    db.execute($query);
  - Suppose:

    $userName = 'blackhat'

    $itemName = “ name' OR 'a'='a “
  - Then the query is: SELECT * FROM items WHERE owner='blackhat' AND itemname='name' OR 'a'='a' ;
  - Semantically equivalent to: SELECT * FROM items

Avoiding SQL Injection

- Validate and sanitize your input.
- Use prepared statements and bind variables.
- Instead of the previous slide's disaster, try:

```
$query = “SELECT * FROM items WHERE owner=? and itemname=?”;
$sth->prepare($query);
$sth->execute($username, $itemname);
```

- Parameterized stored procedures can help as well.
- https://www.owasp.org - TONS of good material here.

Don't Forget the Meatware

- A rogue admin or a disgruntled employee can be your worst nightmare.
- Following our guiding principles can help tremendously. Examples:
  - Don't give all the access to one person.
  - Terminate access immediately upon departure.
  - Multi-factor authentication
- Opinion: Background checks and credit checks are security theatre. Formal, written policies are not.

Running MySQL Securely

- All of the standard system/network security rules apply to a MySQL server.
- Use named user accounts with least privilege.
- Don't allow root to go passwordless, even for connections over the local socket.
- Run the “mysql_secure_installation” script immediately after setup.
- On a server with multiple network interfaces, bind mysqld only to the one you need via the bind-address configuration directive.

Running MySQL Securely: 2

- Disable the old password algorithm via the “old_passwords = 0” configuration directive.
- Don't do this:

  CREATE USER 'foo'@'bar' IDENITIFIED BY 'baz';
- Do this instead:

  CREATE USER 'foo'@'bar' IDENTIFIED BY PASSWORD '*E52096EF8EB0240275A7FE9E069101C33F98CF07';
- Why? It keeps the plaintext password out of .mysql_history and the binary log. Run “SELECT PASSWORD('baz');” on some other machine to get the password hash.

Running MySQL Securely: 3

- Encrypt sensitive data. This can be done at the MySQL layer or in the application (or both?!):
  - If your app servers are separate from your DB servers, and you encrypt at the application layer, the would-be attacker may have to break into two types of machine to retrieve the data. Remember, no easy targets.
- Don't forget about your backups.
- Better yet, don't store any sensitive data. Outsource the processing [and liability] to someone else.
- Full-disk encryption (e.g., DM-CRYPT/LUKS) is good for laptops, but of questionable utility in the server space.

Running MySQL Securely: 4

- If you're trying to use SSL for replication or client connections, you may run into problems if using different software versions.
  - In particular, the Oracle MySQL releases use yaSSL; older versions of Percona Server use yaSSL, newer versions use openSSL. There have been reports of software built with yaSSL having issues connecting to openSSL software and vice versa.
  - Each situation is likely to be different, but if running recent 5.5 / 5.6 builds, one quick solution may be to install the openssl098e compatibility package.
- http://dev.mysql.com/doc/refman/5.5/en/ssl-connections.html
- http://dev.mysql.com/doc/refman/5.5/en/replication-solutions-ssl.html

Quick n' Dirty SSL Replication: 1

- Create your own CA
- Create a server key, then a CSR for that key, and use the CA from step 1 to sign it.
  [see, for example: http://pages.cs.wisc.edu/~zmiller/ca-howto/]
- Put the CA certificate file, server key file, and signed server certificate file on both master and slave. For our purposes, we'll assume the filenames are ca.crt, mysql.key, and mysql.crt.
- Add the following to /etc/my.cnf on both boxes and bounce mysqld:

```
ssl
ssl_ca = /path/to/ca.crt
ssl_cert = /path/to/mysql.crt
ssl_key = /path/to/mysql.key
```

- Create a user that requires a valid X509 certificate:

```
GRANT REPLICATION SLAVE ON *.* TO 'sslrepl'@'%' REQUIRE X509;
```

Quick n' Dirty SSL Replication: 2

- Run the appropriate CHANGE MASTER:
  mysql> change master to master_host='10.10.10.61', master_user='sslrepl', master_log_file='mysqlbin.000006', master_log_pos=554, master_ssl=1, master_ssl_ca='/var/lib/mysql/ca.crt', master_ssl_cert='/var/lib/mysql/mysql.crt', master_ssl_key='/var/lib/mysq;/mysql.key', master_port=3306, master_ssl_cipher='AES128-SHA';
- Start the slave.
- There are other ways to restrict the connection to a specific certificate or a specific CA:

  http://dev.mysql.com/doc/refman/5.5/en/grant.html

Running MySQL Securely: 5

- Commercial versions of MySQL 5.5.16+ have the PAM authentication plugin. Percona also offers a compatible open source version. This can be a good way to manage MySQL access for large numbers of users.
- http://dev.mysql.com/doc/refman/5.5/en/pam-authentication-plugin.html
- http://www.percona.com/doc/percona-pam-for-mysql/intro.html

What's New in MySQL 5.6 ?

- Lots!
  - Connections using passwords created with the old algorithm are disallowed by default. This can be disabled by setting “skip_secure_auth” in my.cnf
  - A fresh RPM install creates a random initial root password rather than leaving it blank – and disallows you from doing anything until you change it with SET PASSWORD. The initial password is stored in .mysql_secret
    http://www.mysqlperformanceblog.com/2013/08/17/mysql-5-6-security-vs-ease-of-use/
  - If you're not doing an RPM install, running mysql_install_db with the “--random-passwords” flag will accomplish the same thing.
  - Passwords don't appear in plaintext in any of the server logs.
  - Commands which involve passwords (including CHANGE MASTER) aren't logged to .mysql_history.

What's New in MySQL 5.6: 2

- SHA256 is available as a new password-hashing algorithm.
  [mysqld]
  default-authentication-plugin=sha256_password
- Using accounts with SHA256 passwords is a bit more complicated. The connection has to either be over SSL or with RSA encryption of the plaintext password.
  http://dev.mysql.com/doc/refman/5.6/en/sha256-authentication-plugin.html
- A password quality/validation plugin is available which can check length of passwords, presence of numbers or special characters, or match proposed passwords against a dictionary file. This only works if using the PASSWORD() function; setting the password to the hashed value doesn't trigger the plugin.
  http://dev.mysql.com/doc/refman/5.6/en/validate-password-plugin.html

What's New in MySQL 5.6: 3

- MySQL 5.6 can store replication credentials in a table rather than in the traditional master.info file. This is part of the “crash-safe” replication feature, but it's also a small (very small, really) security improvement.

http://dev.mysql.com/doc/refman/5.6/en/replication-options-slave.html#sysvar_master_info_repository

http://dev.mysql.com/doc/refman/5.6/en/replication-options-slave.html#sysvar_relay_log_info_repository

Parting Shots

- Much of security is mindset and process.
- Don't go overboard. Risk analysis is key.
- Your overall system security is only as good as its most insecure part. Trust no one, Agent Mulder.
- MySQL can be secured reasonably well by following good practices and making use of all the tools available.
- Any Questions?

Email: ernest.souhrada@percona.com
Twitter: @denshikarasu