Database Security

- Basic terms
- SQL query basics
- Database Management and hardening
  - Connecting to DB
  - User accounts and authentication
  - Storing credentials
  - Permissions
  - Configuration
- SQL Injection basics

Relational Database Systems

- Data in structured form in related tables
- Fixed schema of tables
- Structured Query Language (SQL)
- Different DBMS (Database Management System) technologies
  - MySQL/MariaDB, PostgreSQL, MSSQL, Oracle, …
- Deployment
  - Same server as web application (small applications, test)
  - Dedicated DB server, or multiple servers (scalability, high availability)
- Non-relational databases: NoSQL (unstructured, Big Data)

Structure of SQL Database

- Database
- Table (relation)
- Columns
- Special databases
  - Information about databases, tables, columns, permissions, etc.
  - E.g. INFORMATION_SCHEMA in MySQL

test.users

id	username	password
1	admin	admin123
2	alice	Pa55w0rd
3	bob	superman

SQL Queries – CREATE

- Database creation

```
CREATE [OR REPLACE] {DATABASE | SCHEMA} [IF NOT EXISTS] db_name

CREATE DATABASE test;

USE test; # connect to DB
```

- Table creation

```
CREATE [OR REPLACE] [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name

    (create_definition,...)
```

  - create_definition – columns definition

```
CREATE TABLE IF NOT EXISTS users (id bigint auto_increment primary key, username varchar(32), password
varchar(32));
```

- Database / table removal

```
DROP {DATABASE | TABLE} {db_name | tbl_name}

DROP DATABASE test;
```

SQL Queries – SELECT

id	username	password	email
1	admin	admin123	admin@localhost
2	alice	Pa55w0rd	alice@mail.com
3	bob	superman	bob@mail.eu

```
SELECT * FROM users;

SELECT * FROM users WHERE username = 'admin';

SELECT username,password FROM users WHERE email = 'admin@localhost';

SELECT email FROM users WHERE username LIKE 'a%';
```

Database Management and Hardening

- Connecting to the database
- User accounts and authentication
- Storing credentials
- Permissions
- Configuration
- Note: focus on MariaDB, principles applicable also to other DBMSs

Connecting to the Database

- Locally
  - `mysql -u user -ppassword test # insecure`
  - password is in the history and also in the list of running processes!
  - `mysql -u root -p # prompt for password`
- Remotely
  - `mysql -h 192.168.1.20 -u user -p myapp`

Connecting to the Database - Hardening

- Isolate DB server as much as possible
  - Disable network access (TCP)
  - Listen only on localhost
  - Restrict access to the network port to specific hosts with firewall
  - Separate network segment/DMZ for DB server, isolated from application server
- Restrict access to administration interface (e.g. PHPMyAdmin)
- Configuring encryption for TCP connections
  - TLS, strong ciphers, ...

User Accounts and Authentication

- Creation of user account

```
CREATE USER account_name [authentication_option];
```

- account name
  - ‘user_name’@’host_name’ # allow login from machine with hostname ‘host_name’
  - ‘user_name’@’192.168.1.0/24’ # allow login from network range
  - ‘user_name’ # host_name value will be ’%’ (wildcard – allow all sources)
- authentication_option
  - IDENTIFIED BY ‘password’ # in plaintext, stored in DB hashed with function PASSWORD()
  - IDENTIFIED BY PASSWORD ‘password_hash’
  - IDENTIFIED VIA authentication_plugin
- Examples:

```
CREATE USER ‘user’@’localhost’ IDENTIFIED BY ‘password’;

CREATE USER ‘foo’@’test’ IDENTIFIED VIA pam;
```

User Accounts and Authentication - Hardening

- Require authentication for all accesses (including local)
- Use strong passwords
- Separate user account for each application
- Regular account review
  - Remove unneeded accounts (e.g. decommisioned application, left employee)
  - Permissions

Permissions

- Grant permissions

```
GRANT priv_type ON priv_level TO username;
```

- priv_type
  - ALL PRIVILEGES
  - Global privileges (DB administration, user administration, etc.)
    - e.g. `SHOW DATABASES, SHUTDOWN, GRANT OPTION, CREATE USER,...`
  - Database privileges (working with specific database)
    - e.g. `CREATE, DROP, ...`
  - Table privileges (working with specific table)
    - e.g. `SELECT, CREATE, UPDATE, DROP, ...`
- priv_level
  - `*.*` - global privileges
  - `db_name.*` - database privileges
  - `db_name.tbl_name` – table privileges
- Examples:

```
GRANT ALL PRIVILEGES ON test.* TO 'user'@'localhost';

GRANT SELECT, UPDATE, DELETE, INSERT on eshop.items TO 'user'@'localhost';
```

- Show permissions – `SHOW GRANTS;`
- Remove permissions – `REVOKE`

Permissions - Hardening

- Use principle of “least privilege”
- Do not use built-in root account
- Do not grant administrative rights over the database instance
- Allow access only from specific hosts
  - localhost, application server(s)
- Only grant the account access to the specific database it needs
- Only grant the required permissions on the databases
  - often it is enough to allow SELECT, UPDATE, DELETE

Storing Credentials

- Credentials to DB
  - Do not store in source code / comments
  - In configuration file outside web root
  - Appropriate permissions so that it can be accessed only by required user(s)
  - Do not store in repositories
- Passwords in DB
  - Do not store in plaintext
  - Use secure hash function and salt
  - Recommended to check against list of leaked passwords

Configuration Hardening

- Underlying OS should be hardened and updated
- Regular installation of security updates
- DB service should run under low privileged user account
- Remove any default accounts and databases
- Regular backups, ideally encrypted
- Enable auditing and review audit records

SQL Injection

- SQL Injection vulnerability arises in application when it incorporates user controllable input into SQL query to the DB
- Inserting metacharacters
- Injecting valid SQL commands

```
$login = $_GET["login"];

$pw = $_GET["pw"];

$query = "SELECT * FROM users WHERE
login='$login' AND password='$pw'";

$result = mysql_query($query);
```

SQL Injection – Example 1

- Manipulate the query using metacharacters (reserved characters in SQL)
  - ‘, “, `,(,), \, --, /*, */, #, %
- Query finishes with an error
  - error message can reveal further information

```
SELECT * FROM users WHERE login='$login' AND
password='$pw'";

http://www.page.sk/login.php?login='

SELECT * FROM users WHERE login='' AND password='$pw'";
```

error: You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near "'" at line 1
client_info: mysqlnd 5.0.12-dev - 20150407 - $Id: 38fea24f2847fa7519001be390c98ae0acafe387 $
host_info: Localhost via UNIX socket

) Query: SELECT username FROM accounts WHERE username='"; (0) [Exception]

SQL Injection – Example 2

- Manipulation of original query by inserting comments

```
SELECT * FROM users WHERE login='$login' AND password='$pw'"

http://www.page.sk/login.php?login=admin'-- &pw=123

SELECT * FROM users WHERE login='admin' -- ' AND
password='123'"
```

Note: Different DMBSs can behave differently when processing comments: e.g. MariaDB/MySQL requires that "--" is followed by space, Oracle and SQL Server do not care.

Note2: When inserting special characters into URL proper encoding is required, e.g. space → +

SQL Injection – Example 3

- Manipulation of original query using tautology

```
SELECT * FROM users WHERE login='$login' AND
password='$pw'
```

```
http://www.page.sk/login.php?login=' OR 1=1 -- &pw=123
```

```
SELECT * FROM users WHERE login='' OR 1=1 -- ' AND
password='123'"
```

Note: When inserting special characters into URL proper encoding is required, e.g. space → +

UNION based SQL Injection

- UNION operator allows merging the results of multiple SELECT queries
- Necessary condition: same number of columns in the output
- Steps:
  1) Identify number of columns
  2) Identify columns in the output
  3) Append desired information

UNION based SQL Injection - Example

myapp.users

id	username	password	email
1	admin	admin123	admin@localhost
2	alice	Pa55w0rd	alice@mail.com
3	bob	superman	bob@mai

myapp.articles

authorId	name	title	text
1	John	Java How-to	...
2	Jack	Cracking passwords	...

- `SELECT * FROM articles WHERE authorId=$aid`
- `SELECT * FROM books WHERE authorId=1 UNION ALL SELECT * FROM users; --`
- `SELECT * FROM books WHERE authorId=1 UNION ALL SELECT 1, 2, 3, 4 FROM users; --`
- `SELECT * FROM books WHERE authorId=1 UNION ALL SELECT 1, username, password, 4 FROM users; --`

authorId	name	title	text
1	John	Java How-to	...
1	admin	admin123	admin@localhost
2	...		

Defence Against SQL Injection

- **Use ORM**
- Prepared statements (parameterized queries)
- Stored procedures
- Allow-list input validation
- Escaping all user supplied input
- Principle of “Least privilege”

References

- SQL Statements
  - https://mariadb.com/kb/en/sql-statements/
- SQL Language Structure
  - https://mariadb.com/kb/en/sql-language-structure/
- Configuring MariaDB with Option Files
  - https://mariadb.com/kb/en/configuring-mariadb-with-option-files/
- Database Security Cheat Sheet
  - https://cheatsheetseries.owasp.org/cheatsheets/Database_Security_Cheat_Sheet.html
- SQL Injection Prevention Cheat Sheet
  - https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html