

Security

- [Be careful with backups of config.php](#)
- [SQLInjection Protection](#)

Be careful with backups of config.php

One big security hole that is all too common with Moodle is to take a backup of the `config.php` file and give it a non-php extension e.g. `config.php.bak`.

What this usually does is leave the file visible for viewing by anyone that visits that url e.g. `https://yourmoodle.com/config.php.bak`

Because the file is not a PHP script, it will be rendered in **plain text** by the browser, and therefore expose any sensitive information in `config.php` like your database name, database user and password!

So if you need to back up your `config.php` file before editing make **sure** you put that backup outside of the public web server directory!

SQLInjection Protection

One of the most important things you can do if you are developing your own Moodle plugins is to ensure that you use the Moodle Data Manipulation Library (DML) when working with the database. This layer contains a number of functions that have already been security reviewed and patched for SQL injection vulnerabilities.

To understand SQL Injection in Moodle, take a look at the dev document [Security:SQL Injection](#).

To summarise, the issue is if parameters sent to database queries are not appropriately cleaned and vetted, malicious SQL could be included in the request. The two important things are:

- Use the DML Library
- Always use place holders for parameters rather than say concatenating input. That is, never, ever *trust* user input.

For example if you do need to execute SQL make sure anything passed to the SQL is through a parameter.

```
global $DB;
$email = // email address from user input for example
```

```
// This would be a really bad idea
$query = "
select email
from {user}
where email = $email";
// Execute SQL in an non-Moodle DML way e.g. mysqli functions
```

What if as a user I specified my email address as:

```
'@'; update mdl_user set email = 'blackhat@hacker.com' where id = 2;
```

Even with quotes around the email to make it a string this isn't hard to get around:

```
$query = "  
[]select email  
[]from {user}  
    where email = '$email'";
```

If we just cater for the quotes in MySQL:

```
@'; update mdl_user set email = 'blackhat@hacker.com' where id = 2;
```

This is why the DML library is better e.g.

```
// This is better  
$query = "  
[]select email  
    from {user}  
    where email = :email  
";  
$params = [  
[]'email' => $email  
];  
$result = $DB->get_record_sql($query, $params);
```

In this scenario, the `get_record_sql` method for the appropriate database platform will check our parameters. For example with MySQL this would be in `./lib/dml/mysql_native_moodle_database.php` (and it uses `get_records_sql()`).

Which calls `fix_sql_params` in `./lib/dml/moodle_database.php`. If there is ever a SQL injection issue, then

this is one place in Moodle it can be fixed which means your code will also be patched with any security patches because you used the DML library instead of writing queries yourself.

Also even if you were to use just `$DB->execute($query)` this will still throw an error about having multiple queries together (the select and the update).

For a list of a methods in the data manipulation library see: [Data manipulation API](#)