

# Objective--Badge Manipulation (Part 3)

## Solution (so far)

Here is the query we will work with:

```
SELECT first_name,last_name,enabled FROM employees WHERE authorized = 1 AND uid = '{}' LIMIT 1
```

The query will return three things to the application. It will return strings with the first and last names, and a value for enabled, likely TRUE or 1 if we want to get in the door. Our modified query must return the same thing: two strings and a 1.

The value on Alabaster's badge must be the uid. (In part 1, we found that the QR code on Alabaster's badge contains "oRfjg5uGHmbduj2m." Our input will replace the curly braces ( {} ). When Alabaster scans his badge, this query is executed:

```
SELECT first_name,last_name,enabled FROM employees WHERE authorized = 1 AND uid = 'oRfjg5uGHmbduj2m' LIMIT 1
```

It searches the employees table and if it finds a row with Alabaster's uid and a value of 1 for authorized, it returns something like Alabaster, Snowball, 1. Alabaster's card has been disabled, so authorized must be set to 0.

A Google search for "mariadb comment" takes us to [this link](#), which shows us:

1. From a '#' to the end of a line:

```
SELECT * FROM users; # This is a comment
```

2. From a '--' to the end of a line. The space after the two dashes is required (as in MySQL).

```
SELECT * FROM users; -- This is a comment
```

## Build the SQLI input

We know that our input, which we can represent by xxxxxx, will make the query look like this:

```
SELECT first_name,last_name,enabled FROM employees WHERE authorized = 1 AND uid = 'xxxxxx' LIMIT 1
```

If we end our input with a comment (xxxxxx#) we will have this, where the blue text is a comment:

```
SELECT first_name,last_name,enabled FROM employees WHERE authorized = 1 AND uid = 'xxxxxx#' LIMIT 1
```

Now we have a problem. Our input belongs to the uid statement, and we may get an error because we removed a single quote and the remaining quote no longer matches. So, let's begin our input with a single quote to close out the first single quote after uid=. Now we have 'xxxxxx# and the query looks like this.

```
SELECT first_name,last_name,enabled FROM employees WHERE authorized = 1 AND uid = ''xxxxxx#' LIMIT 1
```

All we (i.e., you) need to do now is fill in the xxxxxxxx.

## Hand In

We need to add something that will overwrite the values since the first part, `SELECT first_name,last_name,enabled FROM employees WHERE authorized = 1 AND uid = ''` will execute no matter what we do. One way to make the query return our data instead of data from the table is to use either `UNION` or `UNION ALL`. In the [link to OWASP](#) that Pepper gave us, look at the paragraph “An example of signature bypass.” A link that I found helpful was from [Netsparker](#), especially the paragraph about “Bypassing second MD5 hash check login screens.” Notice in that example that the input to the new `SELECT` statement includes single quotes, `SELECT 'admin'` instead of `SELECT admin`. The one with the quotes will just return the string, `admin` in this case. The one without the quotes will return values for the variable `admin`.

- 1) What is the SQLI that opens the door for you?

