**On SQL and SQL injections**

Generally, how do we store the data that we want to use most efficiently? Many different database designs have been created as more efficient ways to store useful things, and many of these databases use SQL (structured query language) to access that data.

Why do we care about SQL with regards to cybersecurity? The answer is that, because SQL databases are widely used to store sensitive information such as user login credentials, it's possible to access credentials we weren't supposed to by using the right SQL commands. For example, say our web application takes in a **username** and **password**, then queries user accounts stored in a database called **wicys_db** with a table called **users** containing the following:

SELECT * FROM **users** WHERE **username**='dan' AND **password**='mypassword'

Seems fine, right? Something to note about SQL are the single quotes which must surround strings to separate them from the actual query we may be trying to execute.

So, would it be possible to "escape" the string that the SQL query assumes is the password? You could potentially add more quotation marks to your user input if the web application you're using hasn't protected against it. This would look something like this:

SELECT * FROM **users** WHERE **username**='Dan' AND **password**=' ' OR _____'

See how that closes off the input for password and gives us free reign to do whatever we want with the other condition? Now, we can inject whatever SQL we want in that OR condition. Some may also like to finish off their current statement with a semicolon then move on to a task like deleting someone's database.

For the scope of solving the fridge problem in this game, notice that we know our username but can't remember our password. So, we need to find some OR condition that makes this query return our account. Notice that, to compare if something is true in SQL, it only takes one equals sign. So, checking if something is equal would be done with A=B.

Think about a condition you can satisfy which is always true, and think about how you can form your input so that the final string enclosure is matched with another starting string quote. Try to solve this problem with the given hints, but if you're stuck, there's no shame in looking it up!

**More info on SQL's SELECT statement**

To query from any database, as you may guess from the above example, one would structure their query as such:

**Command Syntax:** SELECT [something] FROM [database_name] WHERE [condition]

Here, the "something" may be a field defined in a database. As SQL has the word "structured" in the name, it does only operate on data for which the structure is predefined. So, our **users** table in the **wicys_db** database may have two attributes, password and username. Maybe it also stores email. Data types must also be specified, so we know those are all strings before writing a query to it.

**SQL in action: an example with starter code**

```
CREATE DATABASE wicys_db;
USE wicys_db;
CREATE TABLE users (
    username varchar(255),
    password varchar(255),
    email varchar(255)
);

INSERT INTO users VALUES ("dan", "mypassword", "dan@illinois.edu");
INSERT INTO users VALUES ("tianyun", "mycoolerpassword", "tianyun@illinois.edu");
INSERT INTO users VALUES ("aishee", "myevencoolerpassword", "aishee@illinois.edu");
INSERT INTO users VALUES ("megan", "mycoolestpassword", "megan@illinois.edu");
```

Shown above is some starting SQL code which we've used to create our **users** table storing three attributes: username, password, and email. After that, the INSERT INTO commands show how we actually put data into a particular database. Note that you can create multiple tables inside of one database; if you're looking to learn more about tables vs. databases, we suggest looking at other general intro to SQL resources.

Going back to the SELECT command, now that we have a table with some data in it, we can ask the database for some data! With the above database and table, we can now try the following command:

**Query:** SELECT * FROM **users**;

And, shown below is the result of that query - we executed all of the above SQL on MySQL to produce the below screenshots. We are asking for all fields from the database table with no filtering conditions, so the output is everything this database table has to offer.

| username | password | email |
|---|---|---|
| dan | mypassword | dan@illinois.edu |
| tianyun | mycoolerpassword | tianyun@illinois.edu |
| aishee | myevencoolerpassword | aishee@illinois.edu |
| megan | mycoolestpassword | megan@illinois.edu |

Now, though, maybe we are only looking for a specific username and password. To query a user from the **users** table with the username "tianyun" and password "mycoolerpassword", we could use the following statement:

**Query:** SELECT * FROM **users** WHERE **username**='tianyun' AND **password**='mycoolerpassword'

Executing this query on the **users** table produces the following result:

| username | password | email |
|---|---|---|
| tianyun | mycoolerpassword | tianyun@illinois.edu |

Note that our query starts with "SELECT *" rather than specifying a value we want such as password. The star denotes all of the fields stored in a particular database, so we will get multiple things in our result. If you would prefer to only select one field such as username, replace the '*' with username. You could also select multiple fields in this manner, eg. "SELECT username, password", then proceed as usual with the rest of the statement.

**A database side note**

One side note on databases, but which is not specific to SQL - databases can also be unsecured through methods other than queries. For example, if your database is publicly accessible to anyone on the Internet who finds the right location, then that may be an issue. Even if your database requires a login to query, it's common for users to configure their databases with default or very easy-to-guess passwords, making them quite easy to break into.

**For more information**

We glossed over the basics of SQL to make this intro to SQL injection brief, but you will be able to craft better attacks (or defenses) with more general knowledge of SQL! If you want to learn more about SQL on your own, the below links could be great places to start.

https://www.khanacademy.org/computing/computer-programming/sql

https://www.w3schools.com/sql/sql_intro.asp

https://www.hackerrank.com/domains/sql