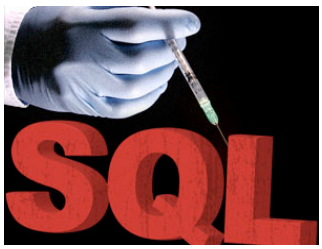
SQL Injection

- What is SQL Injection?
- How Does an Attack Work?
- Mitigating SQL Injection

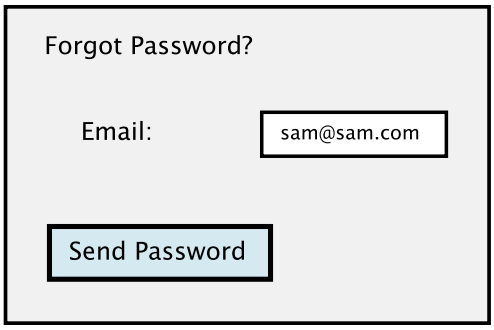
What is SQL Injection?

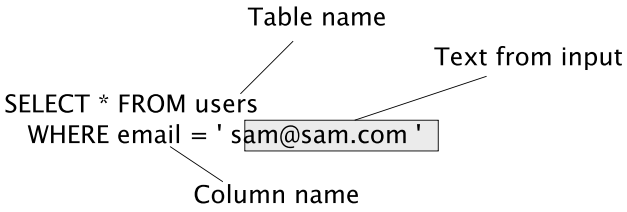
- **SQL injection** is an attack in which an attacker attempts to compromise an application by entering specially crafted SQL syntax into input forms
- Applications are vulnerable to SQL injection if they fail to sanitize user input



How Does an Attack Work?

- Suppose an application simply takes user input and uses it to build an SQL string
- In this example, the input is an email address - note the single quotes that surround characters in SQL commands



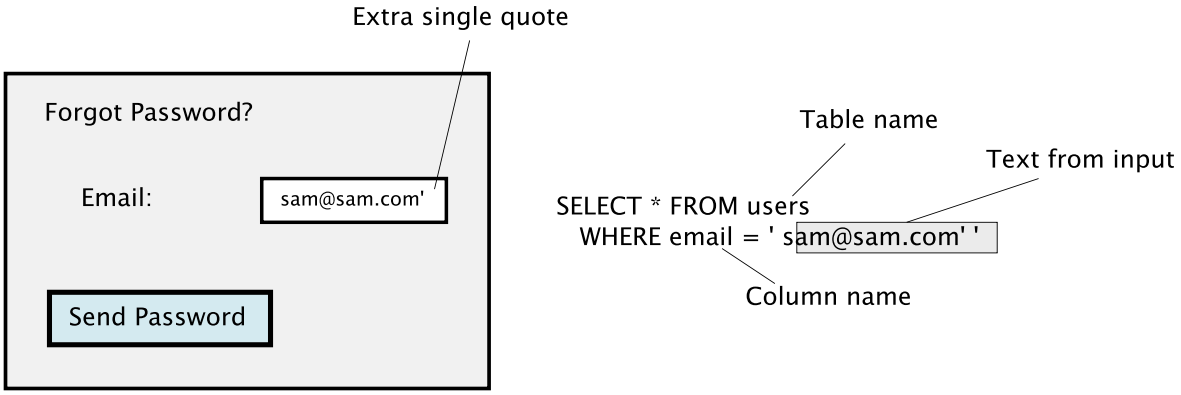


See the following article for a good tutorial on how injection works:

http://www.unixwiz.net/techtips/sql-injection.html

How Does an Attack Work, cont'd

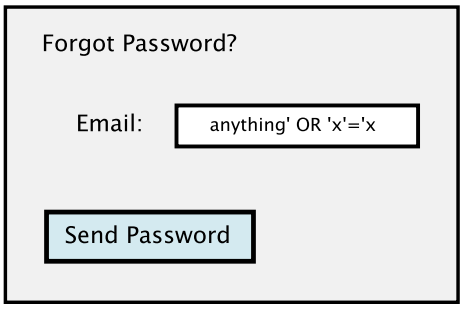
- To start, an attacker can append an extra single quote, leading to illegal SQL syntax
- If application shows an error message such as Server Error 500, attacker knows that the app doesn't sanitize input and thus is vulnerable

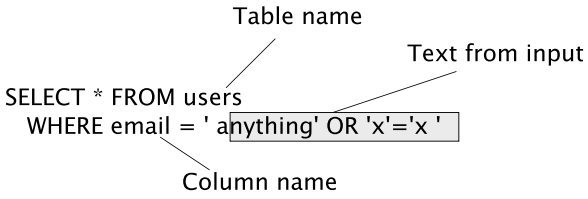


A vulnerable application simply concatenates whatever the user enters to create an SQL command, including the extra single quote. Then if the application has poor error handling, the end user will see an Error 500.

How Does an Attack Work, cont'd

- Another technique is to use the SQL *AND* and *OR* operators to trick the application into executing multiple SQL statements
- In this example, the attacker appends an *OR* that causes the entire SQL expression to always evaluate to TRUE





This example causes the database to return ALL email addresses, since the SQL expression always evaluates to true.

The application, which likely is expecting only at most a single email, would probably send a password to the first one in the list of all email addresses returned by the query.

Another technique is to separate multiple commands with semicolons.

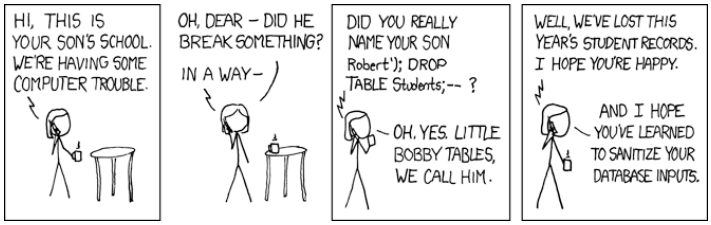
What Can an Attacker Do?

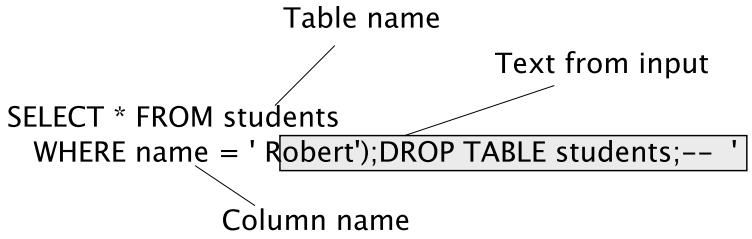
- Using these techniques, attackers can use SQL injection to:
    - Find the names of columns in application's database
    - Find the names of tables in application's database
    - Execute arbitrary SQL commands, including deleting rows and even tables

To accomplish all of this, the attacker may need to do a bit of guessing. For example, the attacker could try to access a column named 'email', and if that doesn't result in a Server Error 500, the attacker knows that the database has a column with that name.

What Can an Attacker Do, cont'd

http://xkcd.com/327/





The XKCD Web comic has an interesting take on SQL injection.

The semicolon acts as an SQL command separator, and allows you to specify multiple commands in a single line.

The -- sequence at the end of the input starts an SQL comment, and is an effective way to "eat" the final quote to avoid an SQL syntax error.

Note that in this case, the XKCD command won't actually work since the ')' character would be invalid.

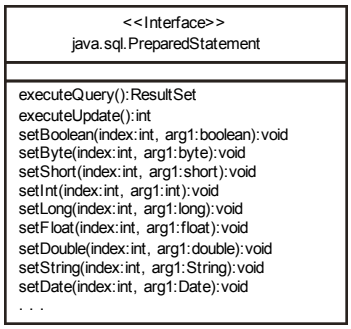
Mitigating SQL Injection

- The best way to avoid SQL injection is to not trust user input when executing SQL based on user interaction
- Strategies:
  - Application sanitizes input by filtering
  - Use prepared statements, which themselves sanitize input
  - Use stored procedures
  - Use an OR/M framework such as Hiberate, JPA or iBATIS, which automatically sanitize SQL

Writing your own filtering code turns out to be quite tricky, especially if you consider that users can insert Unicode characters. For that reason, most Java experts recommend using one of the other strategies listed here.

JDBC Prepared Statements

- JDBC provides the **Statement** and **PreparedStatement** types
- Using prepared statements can increase performance, since they are precompiled
- You typically precompile the statement and then use it multiple times, specifying different parameters each time
- Prepared statements are also less susceptible to **SQL injection** attacks



PreparedStatements are especially useful if you plan to use the same sort of SQL command multiple times. With a PreparedStatement, you write the basic SQL command with placeholders for arguments that you can substitute before you execute the command.

Like most things in JDBC, the performance benefit of PreparedStatements is quite dependent on the database in question and its JDBC driver. But potentially, your database operations can run more efficiently if you use PreparedStatements instead of simple Statements. You should benchmark with your own access patterns to choose which approach provides the best performance for your application.

JDBC Prepared Statements, cont'd

```
1    int ID = 12; double gpa = 3.87; String name = "Lynn";
2
3    PreparedStatement ps = null;
4    try
5    {
6        String cmd =
7            "UPDATE student SET GPA=?, " +
8            "NAME=? WHERE studentID = ?";
9
10       ps = con.prepareStatement ( cmd );
11       ps.setDouble ( 1, gpa );
12       ps.setString ( 2, name );
13       ps.setInt ( 3, ID );
14
15       int rows = ps.executeUpdate();
16   }
17   . . .
```


Notice that when you use Prepared Statements, you do NOT concatenate OR specify single quotes for character data. This, along with the fact that the driver sanitizes input, makes using Prepared Statements a best practice to mitigate SQL injection.

Chapter Summary

In this chapter, you learned:

- What SQL injection is all about
- How to mitigate SQL injection in JDBC applications