User Input

User Input

We have to assume anything that could potentially come from a user might be malicious

There are 2 primary types of attacks we have to be aware of:

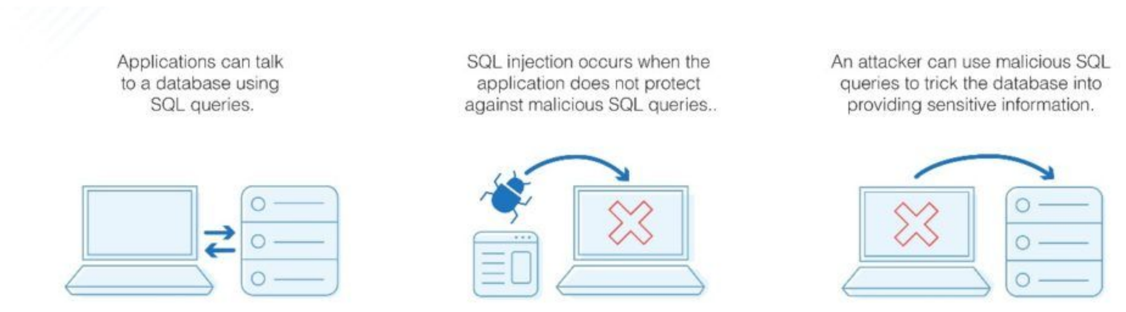
- SQL Injection
- XSS

Cross Site Scripting

- A Cross Site Scripting, or XSS, attack is one in which malicious javascript is inflicted on your users
- Can occur if we display user input on our pages
- Be aware whenever we display information on our page that comes from a user
- Use JSTL library's out tag for this!

```
<h2>Showing Results for <c:out value="${searchTerm}" </h2>
```

SQL Injections


Applications can talk to a database using SQL queries.
SQL injection occurs when the application does not protect against malicious SQL queries..
An attacker can use malicious SQL queries to trick the database into providing sensitive information.

https://www.dnsstuff.com/sql-injection

```
// Somewhere inside a servlet

String searchTerm = request.getParameter("searchTerm");



// DON'T DO THIS!!!

String sql = "SELECT * FROM products WHERE name LIKE '%" +
searchTerm + "%'";
```

Maybe the user searches for a “watch”...

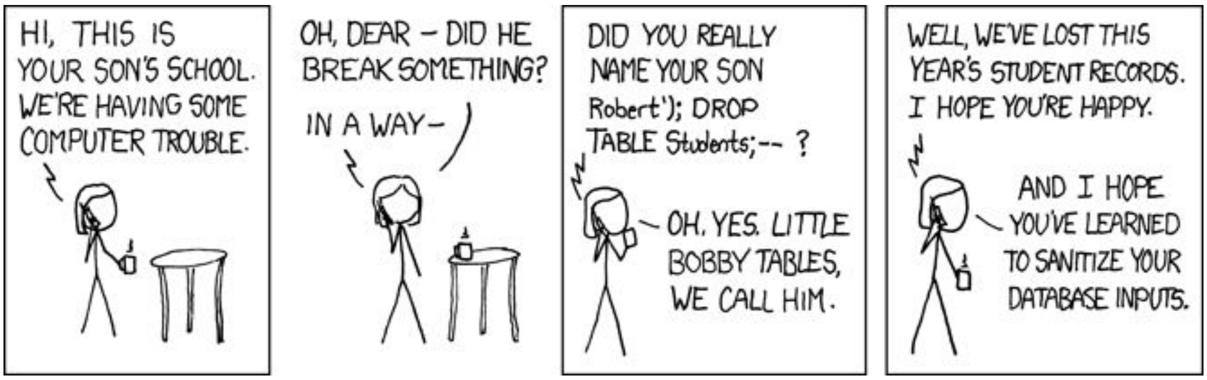
```
SELECT * FROM products
WHERE name LIKE '%watch%';
```

Maybe the user searches for “apple’s watch”...

```
SELECT * FROM products
WHERE name LIKE '%apple's watch%';
```

- Syntax Error
- Application could error out!

SQL Injection


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

https://xkcd.com/327/

Prepared Statements

- Prevents SQL Injection
- The use of placeholders for potentially dangerous values ← User Input
- We will
    1. Define the SQL Query
    2. Bind values to the placeholders

Prepared Statement Methods

- `connection.prepareStatement(sql)`
  a. Create the prepared statement object
  b. when used with INSERT, `Statement.RETURN_GENERATED_KEYS` can be added

- `statement.setXXX(idx, value):`
  a. XXX is the type of the value
  b. idx is the number of the placeholder -- Bind a value to the statement

Prepared Statement Methods

- `statement.executeQuery()`
  a. Execute the prepared SELECT query

- `statement.executeUpdate()`
  a. Execute an INSERT or UPDATE query

Prepared Statement Methods

- `statement.getResultSet`
  a. Get the results of a SELECT query

- `statement.getGeneratedKeys`
  a. Get the generated ids from an INSERT query

```java
String sql = "SELECT * FROM products WHERE name LIKE ?";
String searchTermWithWildcards = "%" + searchTerm + "%";




PreparedStatement stmt = connection.prepareStatement(sql);
stmt.setString(1, searchTermWithWildcards);




ResultSet rs = stmt.executeQuery();
while(rs.next()) {
   // do something with the search results
}
```

```java
String sql = "INSERT INTO products(name, category, price) VALUES (?, ?, ?)";



PreparedStatement stmt =
    connection.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS);



  // For the sake of easier demonstration we are using literals
  here
      stmt.setString(1, "hammer");
      stmt.setString(2, "tools");
      stmt.setFloat(3, 19.99);

      stmt.executeUpdate();
      ResultSet generatedIdResultSet = stmt.getGeneratedKeys();
```