Prepared Statement

Always be prepared

The problem with ordinary Statement

The ordinary Statement was open to SQL injections if fed malicious data.

What would the proper response to that be?

Filter all input to rid all special characters?

There is nothing wrong with always sanitising your inputs. But there is an alternative (which could be used in conjunction with sanitising).

The PreparedStatement class.

SQL injection in bash

Consider a bash script with SQL which makes use of a variable:

```
#!/bin/bash

license=$1

echo "SELECT * FROM cars WHERE
LicenseNumber='"$license"';"|sqlite3 my_cars
```

SQL injection in bash, continued

Where is the vulnerability?

```
#!/bin/bash
license=$1
echo "SELECT * FROM cars WHERE
LicenseNumber='"$license"';"|sqlite3 my_cars
```

SQL injection bash example

We want to fetch one car using the license number as argument. But:

```
$ ./sqlinj.sh "' OR 1=1;--"
```

The argument will actually be:

```
' OR 1=1;--
```

The SQL statement will thus be:

```
SELECT * FROM cars WHERE LicenseNumber='' OR 1=1;--';
```

Escaping using printf “%q”

```
#!/bin/bash

#license=$1
printf -v license "%q" "$1"
echo "Argument: $license"
license=$(echo "$license"| sed -e 's/\\\ /\ /') # unescape spaces

echo "QUERY:"
echo "SELECT * FROM cars WHERE LicenseNumber='"$license"';"
echo "SELECT * FROM cars WHERE LicenseNumber='"$license"';" |sqlite3 my_cars


$ ./sqlinj.sh "' OR 1=1;--"|head
Argument: \'\ OR\ 1=1\;--
QUERY:
SELECT * FROM cars WHERE LicenseNumber='\'\ OR\ 1=1\;--';
Error: near line 1: unrecognized token: "\"   # SQLite3 said that!
```

Solution in Java with PreparedStatement

In Java we should use the PreparedStatement if we are worried about SQL injections.

Show us the code...

The idea behind a prepared statement is to use placeholders and set the corresponding values using safe functions (the functions escape crap input):

```
public int updateHTTPSbyName2(String name, boolean https){
  String sql = "UPDATE municipalities SET HTTPS=? WHERE name= ?";
  int result=0;
  try{
    PreparedStatement pStm = db.preparedStatement(sql);
    pStm.setInt(1, (https?1:0));
    pStm.setString(2,name);
    result=pStm.executeUpdate();
    return result;
  }catch(Exception e){
    System.err.println("Error creating prepared stm: "+e.getMessage());
    return -1;
  }
}
```

Solution in PHP with PreparedStatement

PHP also has the concept of prepared statements.

Consider a database with one table cars:

CREATE TABLE cars(id INTEGER primary key, make text, license text);

And a PHP page which reads make from the GET parameters and searches for cars with matching make in the cars table.

Example result

Got connection: Server version: 3.11.0

Searched for Make: Ford Mondeo

id	make	license
7	Ford Mondeo	CCC 333

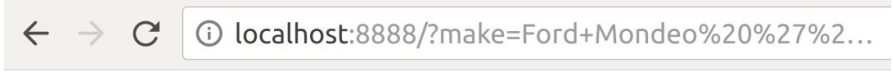
PHP code example

```
$dbh = new PDO($url, $username, $password);
print "Got connection: " .  " Server version: " .
    $dbh->getAttribute(constant("PDO::ATTR_SERVER_VERSION")) . "<br />";
$make = $_GET["make"];
if(!isset($make)){
    $make = "";
}
echo "<p>Searched for Make: $make</p>\n";
echo "<table border=\"1\">";
echo "<tr><th>id</th><th>make</th><th>license</th></tr>";
foreach($dbh->query("SELECT * FROM cars WHERE make = '$make' LIMIT 10") as $row) {
    $id = $row["id"];
    $theMake = $row["make"];
    $license = $row["license"];
    echo "<tr><td>$id</td><td>$theMake</td><td>$license</td></tr>\n";
}
echo "</table>";
```

PHP code example - problem

```
$make = $_GET["make"]; // can be anything! Even SQL!
if(!isset($make)){
    $make = "";
}
echo "<p>Searched for Make: $make</p>\n";
echo "<table border=\"1\">";
echo "<tr><th>id</th><th>make</th><th>license</th></tr>";
foreach($dbh->query("SELECT * FROM cars WHERE make = '$make' LIMIT 10") as $row) {
    $id = $row["id"];
    $theMake = $row["make"];
    $license = $row["license"];
    echo "<tr><td>$id</td><td>$theMake</td><td>$license</td></tr>\n";
}
echo "</table>";
```

Example SQL injection



Got connection: Server version: 3.11.0

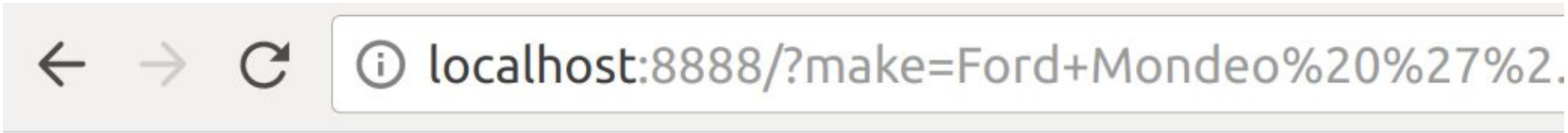
Searched for Make: Ford Mondeo ' or 1=1;--'

id	make	license
1	Honda	AAA 111
2	Honda	ABC 111
3	Volvo	CCC 112
4	Volvo	BBB 112
5	Dodge Challenger	BBC 123
6	Ford Mustang	CCC 666
7	Ford Mondeo	CCC 333
8	Opel Astra	CCF 353

Using prepared statement instead

```
$stmt = $dbh->prepare("SELECT * from cars where make = :make LIMIT 10");
$stmt->bindParam(":make", $make); // escapes all text
$stmt->execute();
$row=null;
while ($row = $stmt->fetch()) {
  $id = $row["id"];
  $theMake = $row["make"];
  $license = $row["license"];
  echo "<tr><td>$id</td><td>$theMake</td><td>$license</td></tr>\n";
    }
echo "</table>";
if ($stmt->rowCount() == 0) {
  echo "<br><b>NO ROWS FOUND</b><br>";
}
```

Using prepared statement instead



Got connection: Server version: 3.11.0

Searched for Make: Ford Mondeo ' or 1=1;--'

id	make	license

NO ROWS FOUND

Read up on PreparedStatement

Read about prepared statements (link at the end of lecture) and experiment with it.

Read...

Java:

http://docs.oracle.com/javase/tutorial/jdbc/basics/prepared.html

PHP.net:

http://php.net/manual/en/pdo.prepare.php