Network & Web Security

SQL Injection Attacks & Countermeasures (Part II)

Professor Travis Peters
CSCI 476 - Computer Security
Spring 2020

Some slides and figures adapted from Wenliang (Kevin) Du's
Computer & Internet Security: A Hands-on Approach (2nd Edition).
Thank you Kevin and all of the others that have contributed to the SEED resources!

Countermeasures: **Filtering & Encoding Data**

- Before mixing user-provided data with code, ***inspect the data & filter out*** any character that may be interpreted as code.
- Special characters are commonly used in SQL Injection attacks. To get rid of them, ***encode them.*** Encoding a special character ***tells parser to treat the encoded character as data and not as code***. E.g.,

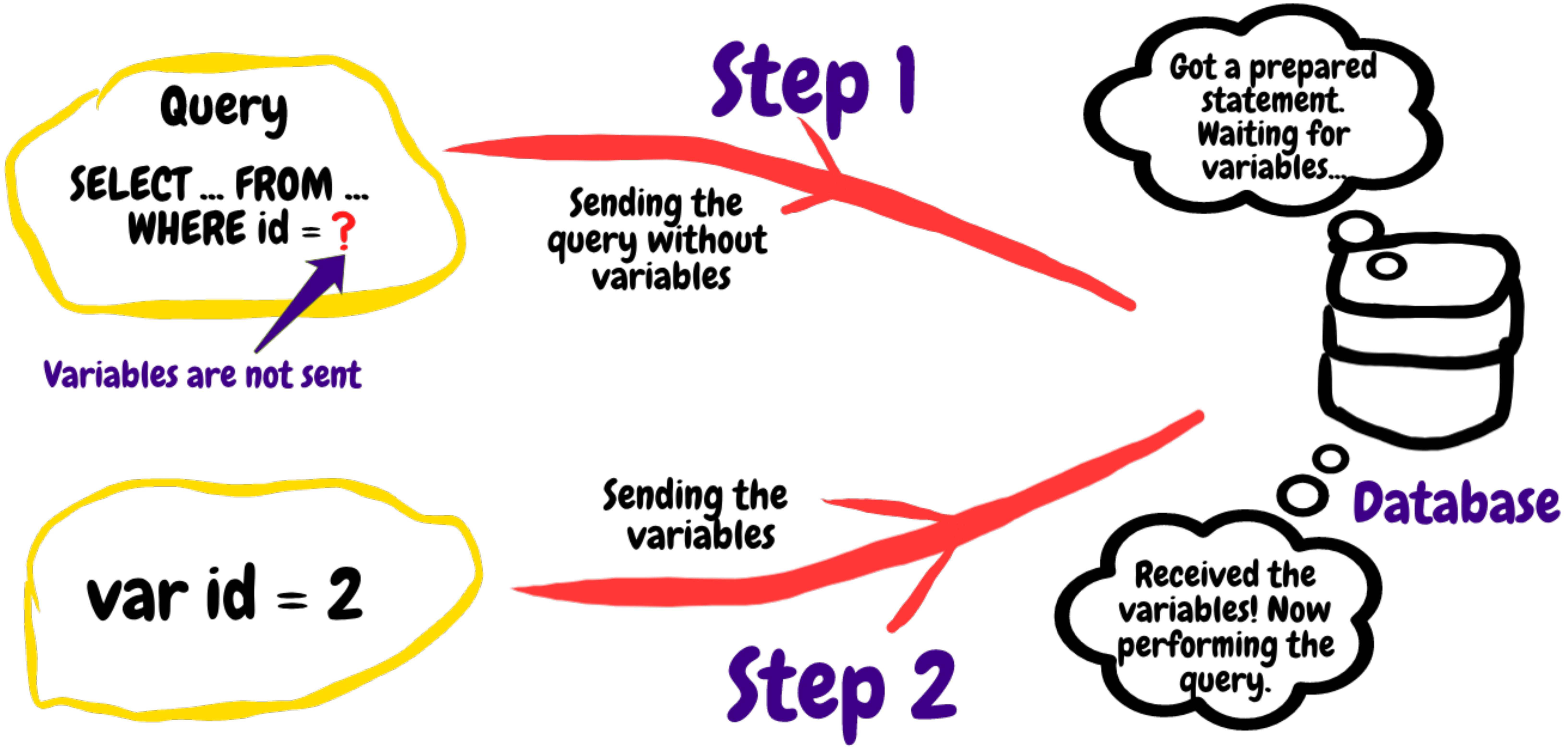
```
Before Encoding:  aaa' OR 1=1 #
 After Encoding:  aaa\' OR 1=1 #
```

- PHP's `mysqli` extension has a built-in method called **`mysqli::real_escape_string()`**, which can be used to encode characters that have special meanings in SQL.
  E.g.,

```
/* getdata_encoding.php */
<?php
   $conn = new mysqli("localhost", "root", "seedubuntu", "dbtest");
   $eid = $mysqli->real_escape_string($_GET['EID']);
   $pwd = $mysqli->real_escape_string($_GET['Password'];
   $sql = "SELECT Name, Salary, SSN
           FROM employee
           WHERE eid= '$eid' and password='$pwd'";
?>
```

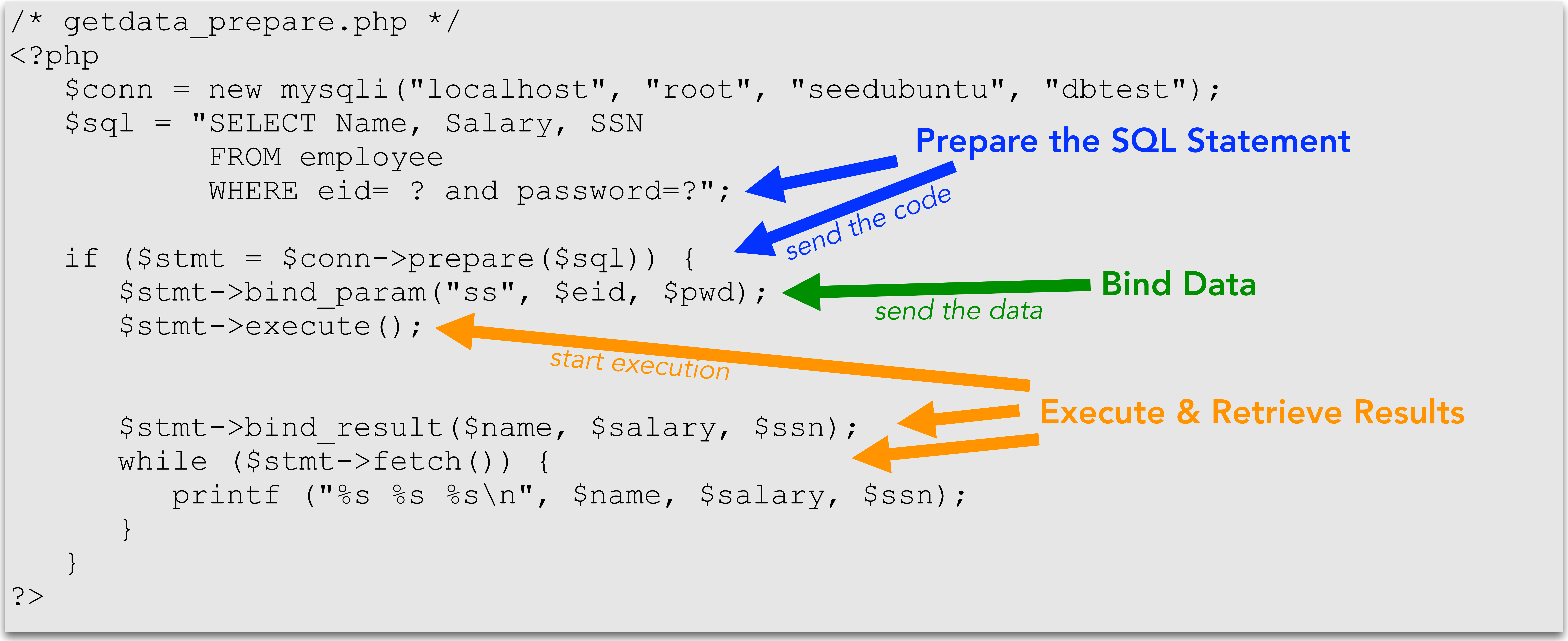
Countermeasures: **Prepared Statements**

- The **fundament cause** of SQL injection attacks is mixing data and code.
- The **fundament solution** is to separate data and code.
- The **main idea behind prepared statements** is to send code and data in separate channels to the database server.


Query
SELECT ... FROM ...
WHERE id = ?
Variables are not sent
Step 1
Sending the query without variables
Got a prepared statement. Waiting for variables...
Database
var id = 2
Sending the variables
Step 2
Received the variables! Now performing the query.

https://cdn.hyvor.com/uploads/developer/prepared-stmt.png

Countermeasures: **Prepared Statements** *(cont.)*



```
/* getdata_prepare.php */
<?php
   $conn = new mysqli("localhost", "root", "seedubuntu", "dbtest");
   $sql = "SELECT Name, Salary, SSN
           FROM employee
           WHERE eid= ? and password=?";

   if ($stmt = $conn->prepare($sql)) {
      $stmt->bind_param("ss", $eid, $pwd);
      $stmt->execute();


      $stmt->bind_result($name, $salary, $ssn);
      while ($stmt->fetch()) {
         printf ("%s %s %s\n", $name, $salary, $ssn);
      }
   }
?>
```

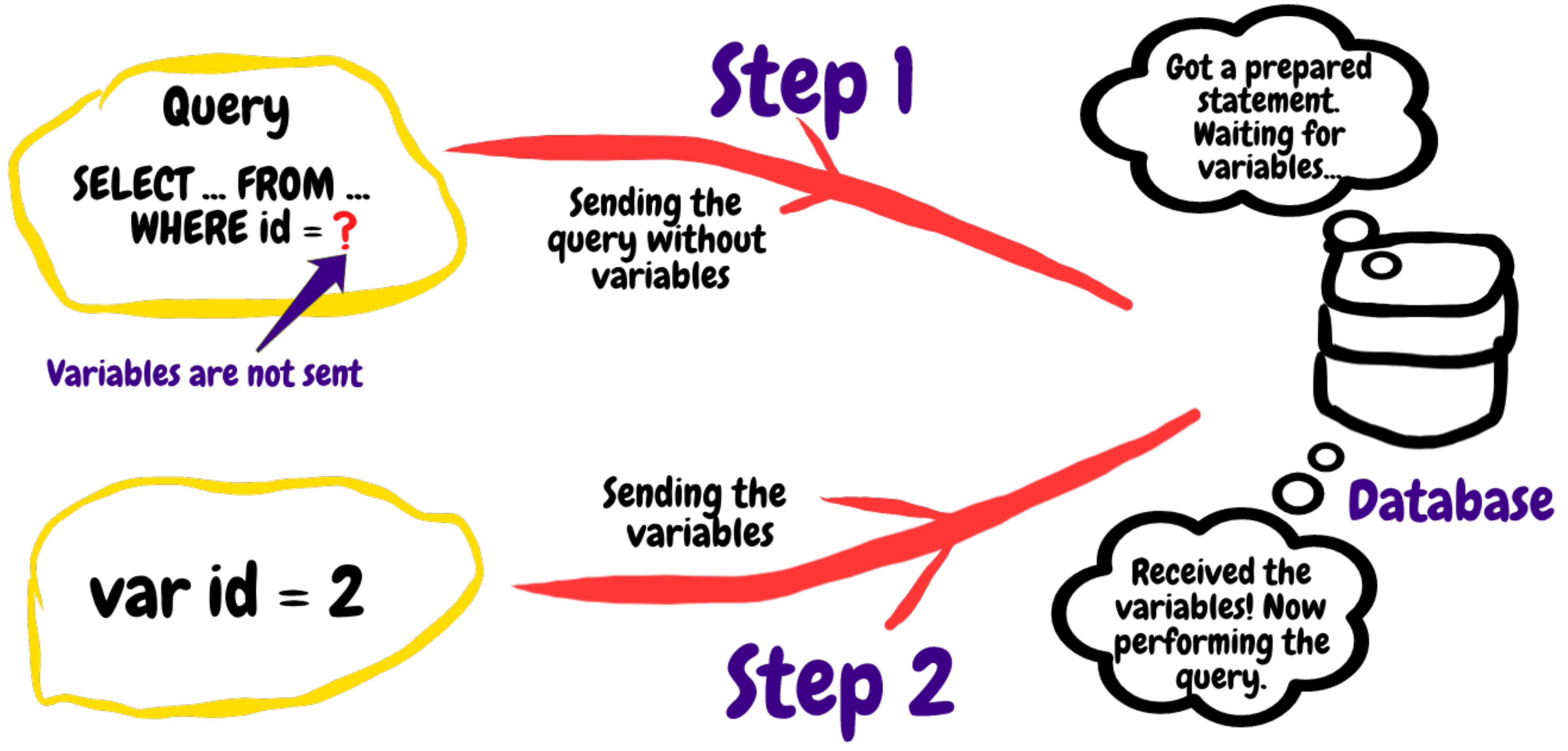
Prepare the SQL Statement
send the code
Bind Data
send the data
start execution
Execute & Retrieve Results

https://cdn.hyvor.com/uploads/developer/prepared-stmt.png

Countermeasures: **Prepared Statements** *(cont.)*

Why Are Prepared Statements Secure?

- Trusted code is sent via a ***code channel***.
- Untrusted user-provided data is sent via ***data channel***.
- Database clearly knows the ***boundary*** between code and data.
- Data received from the ***data channel is not parsed***.
- Attacker can hide code in data, but the code will never be treated as code.


Query
SELECT ... FROM ... WHERE id = ?
Variables are not sent
Step 1
Sending the query without variables
Got a prepared statement. Waiting for variables...
var id = 2
Sending the variables
Step 2
Database
Received the variables! Now performing the query.

Summary

- A brief tutorial of SQL
- The SQL Injection attack w/ examples
- The fundamental cause of the vulnerability
- How to defend against SQL Injection attacks
  - Data Filtering & Encoding
  - Prepared Statements

You Try!

Exam-like problems that you can use for practice!

- The following SQL statement is sent to the database to add a new user to the database, where the content of the `$name` and `$passwd` variables are provided by the user, but the `EID` and `Salary` field are set by the system. How can a malicious employee set his/her salary to a value higher than `80000`?

```
$sql = "INSERT INTO employee (Name, EID, Password, Salary)
        VALUES ('$name', 'EID6000', '$passwd', 80000)";
```

- The following SQL statement is sent to the database, where `$eid` and `$passwd` contain data provided by the user. An attacker wants to try to get the database to run an arbitrary SQL statement. What should the attacker put inside `$eid` or `$passwd` to achieve that goal. Assume that the database does allow multiple statements to be executed.

```
$sql = "SELECT * FROM employee
        WHERE eid='$eid' and password='$passwd'";
```

- To defeat SQL injection attacks, a web application has implemented a filtering scheme at the client side: basically, on the page where users type their data, a filter is implemented using JavaScript. It removes any special character found in the data, such as apostrophe, characters for comments, and keywords reserved for SQL statements. Assume that the filtering logic does it job, and can remove all the code from the data; is this solution able to defeat SQL injection attacks?
- To defeat code injection attacks when a C program needs to invoke an external program, we should not use `system()`; instead, we should use `execve()`. Please describe the similarity between this countermeasure and the prepared statement, which is a counter- measure against SQL injection attacks.