(Advanced) Computer Security!

Network & Web Security

SQL Injection Attacks & Countermeasures

(part II)

Prof. Travis Peters

Montana State University

CS 476/594 – Computer Security

Spring 2021

https://www.traviswpeters.com/cs476

Today

Reminder!
Please update your Slack, GitHub, Zoom
(first/last name, professional photo/background)

- Announcements
  - Lab 04 due Tuesday (3/9)

- Learning Objectives
  - Basics of the Internet -> towards web apps
  - A (brief) SQL tutorial
  - The SQL Injection (SQLi) attack with examples
  - Understanding SQLi vulnerabilities
  - Understanding SQLi countermeasures

Warning:
Never target websites that you do not have explicit permission to "test" against.

All of our work targets webapps deployed in containers that run within your SEED Labs VM

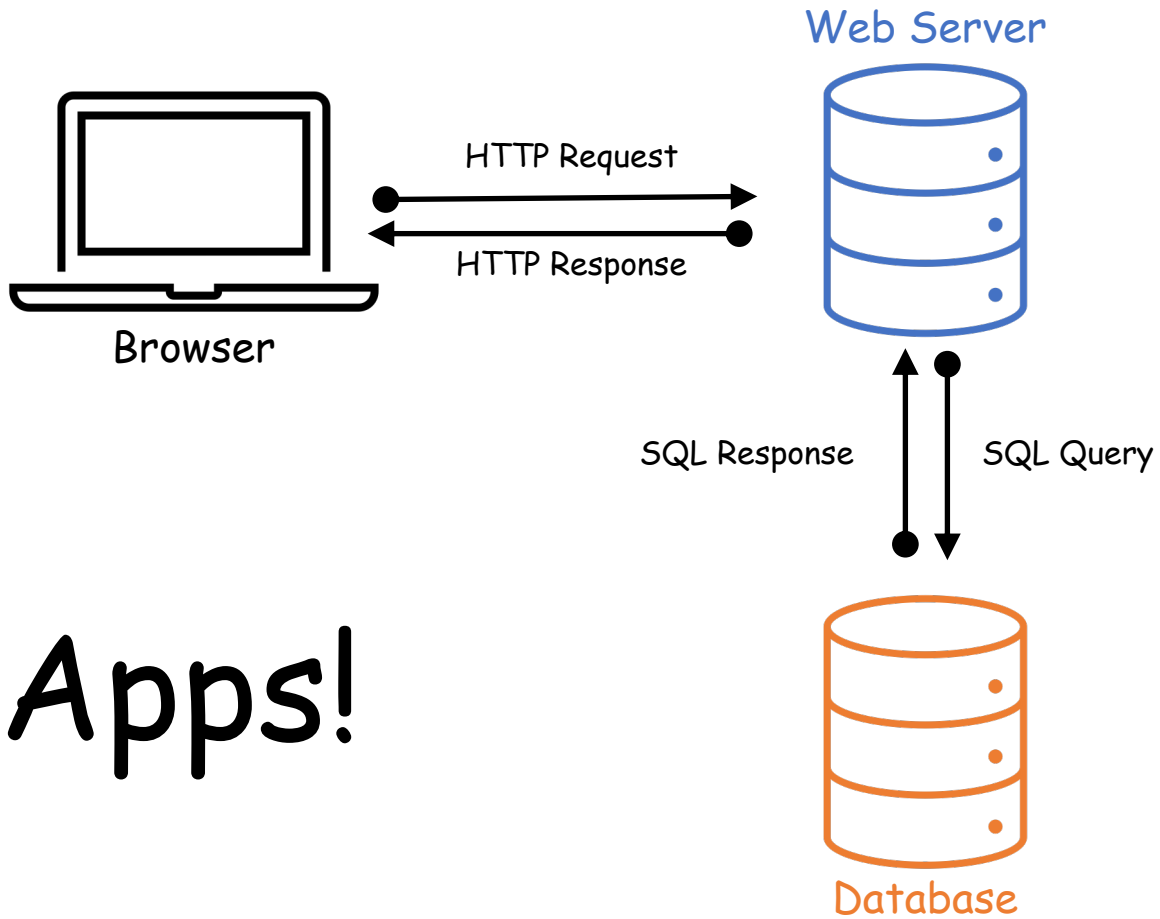
Reflection: Insights from Lab Setup? SQL?

String sanitization

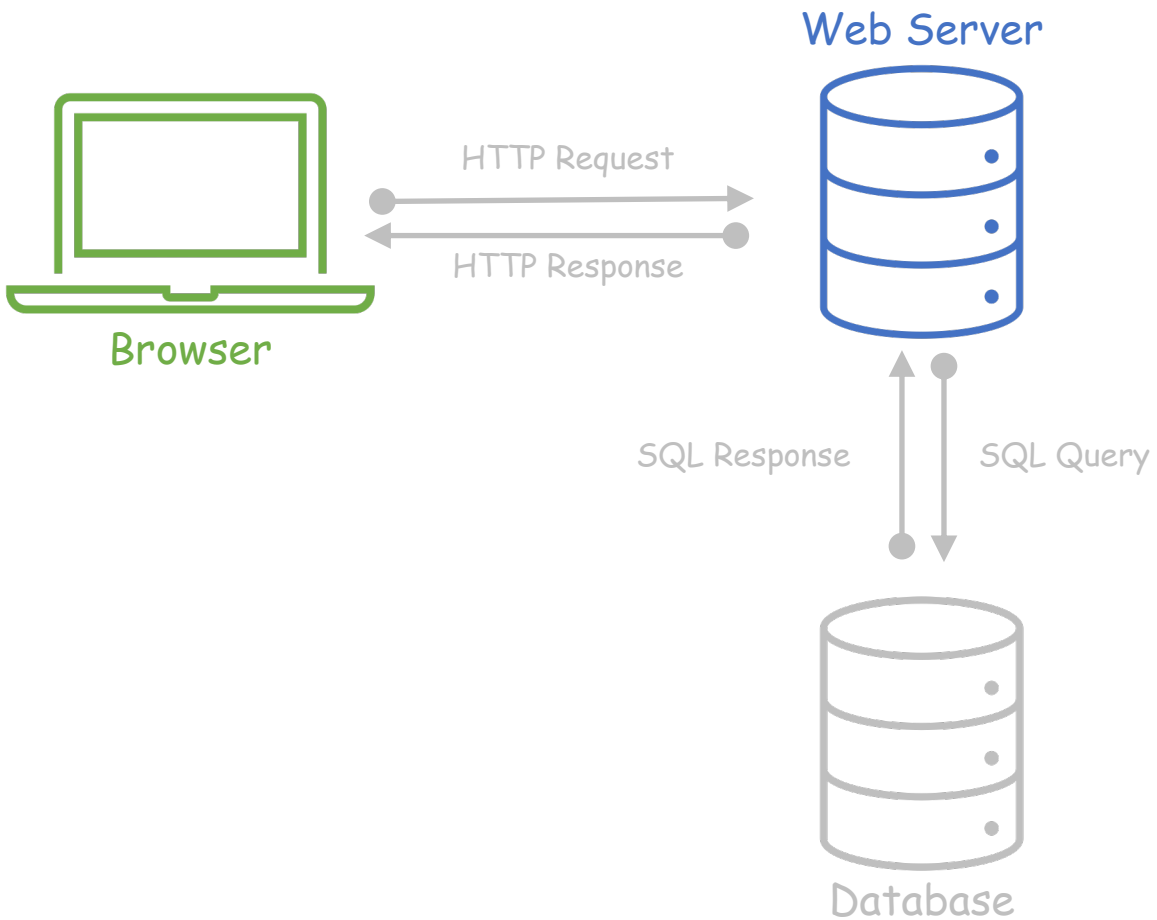
Syntax — crafted payload....
- functions/weaknesses

mixing code & data

Back to Web Apps!



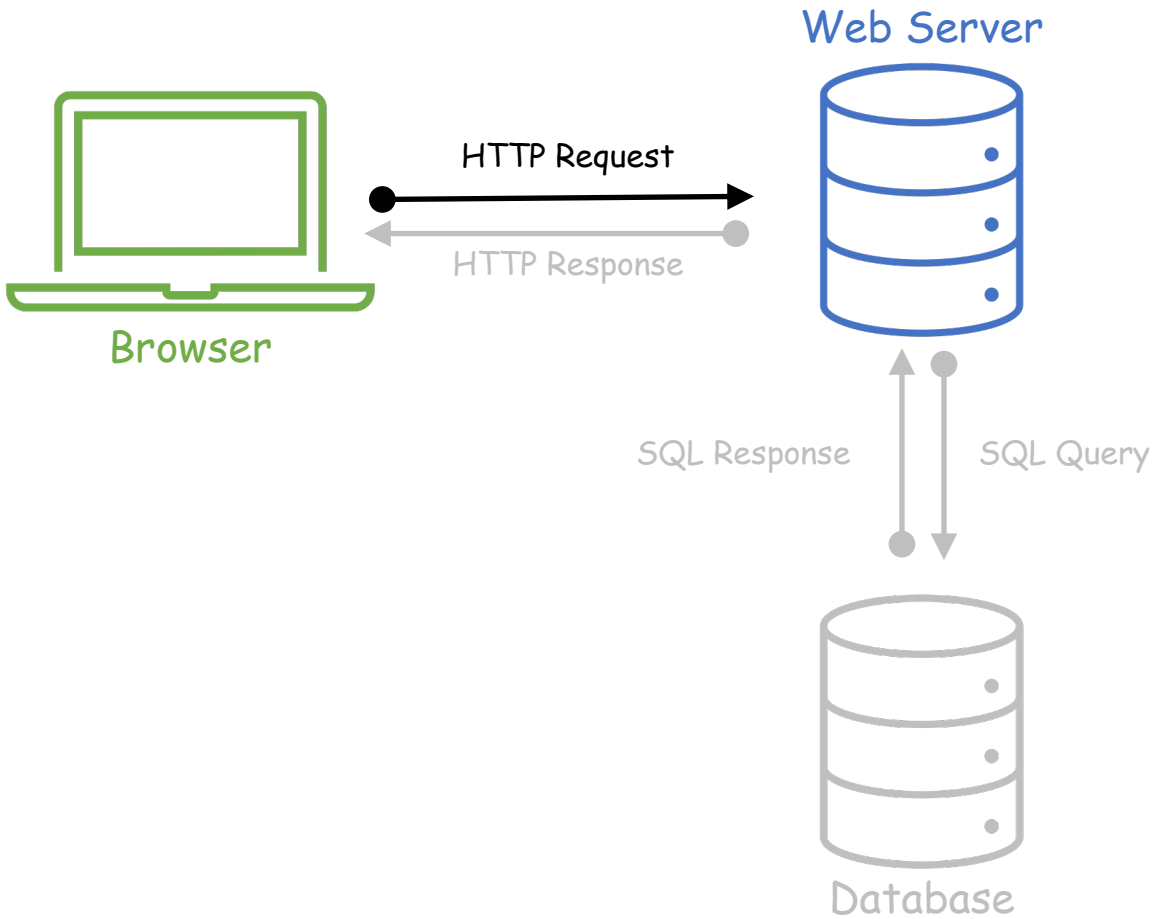
Q: How to get data from a user?



Q: How to get data from a user?

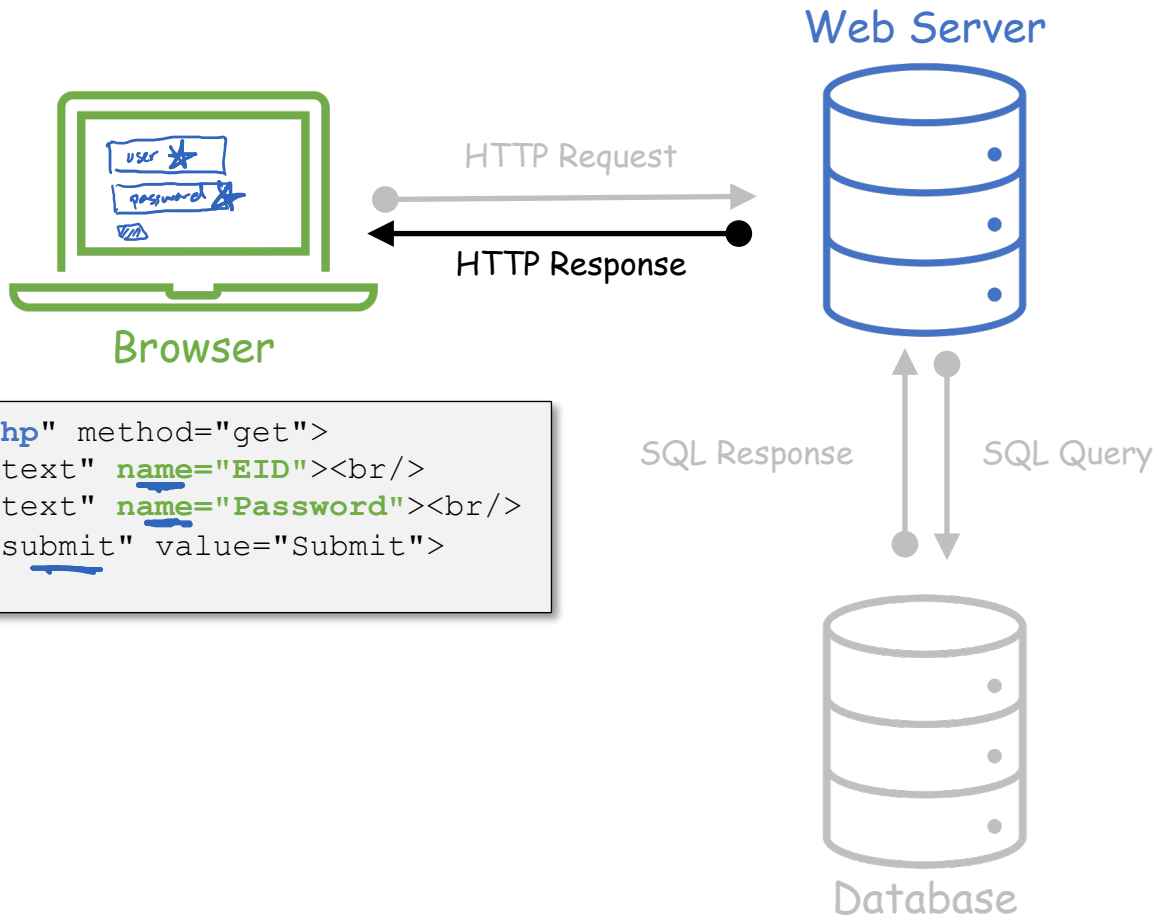
```
http://www.example.com/
```

HTTP Header Live
Burp Suite
(Proxy)



Q: How to get data from a user?

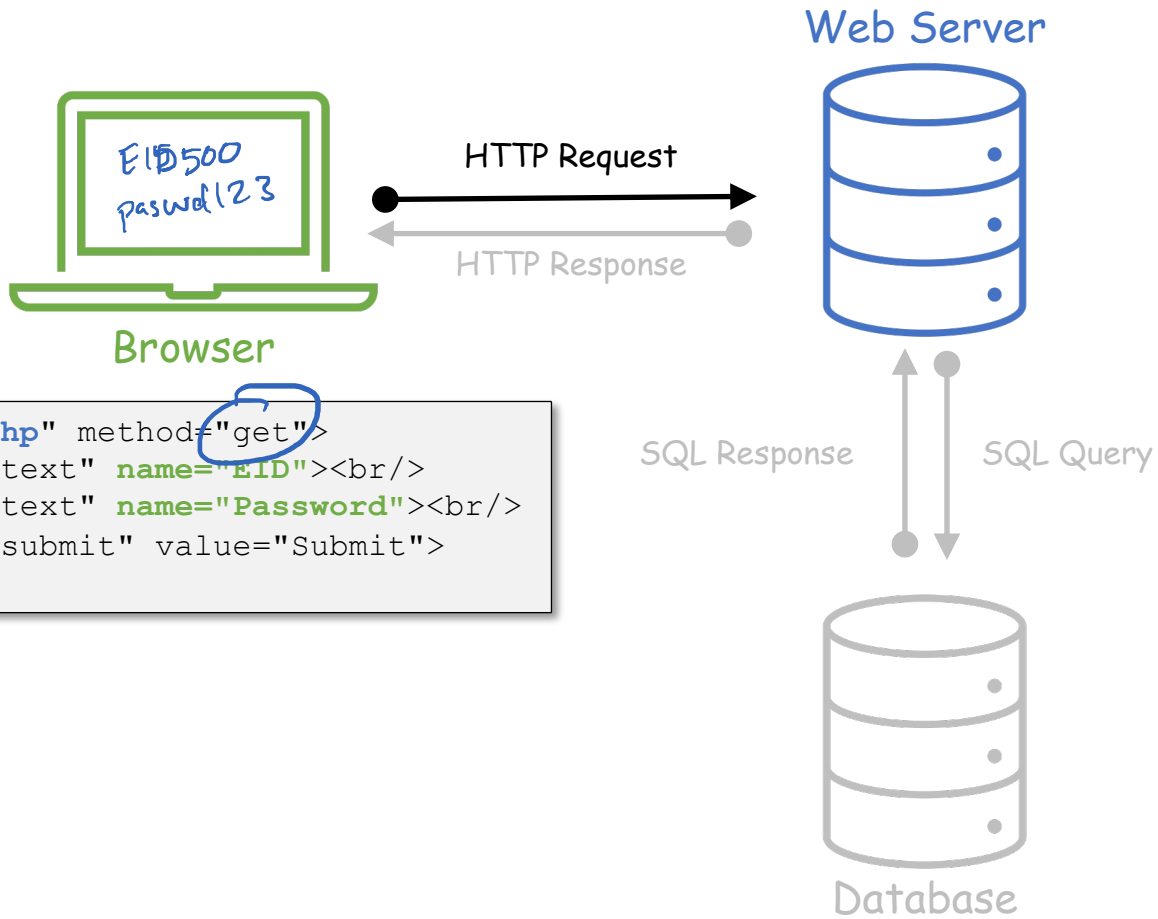
```
http://www.example.com/
```




```
<form action="getdata.php" method="get">
EID:      <input type="text" name="EID"><br/>
Password: <input type="text" name="Password"><br/>
          <input type="submit" value="Submit">
</form>
```

Q: How to get data from a user?

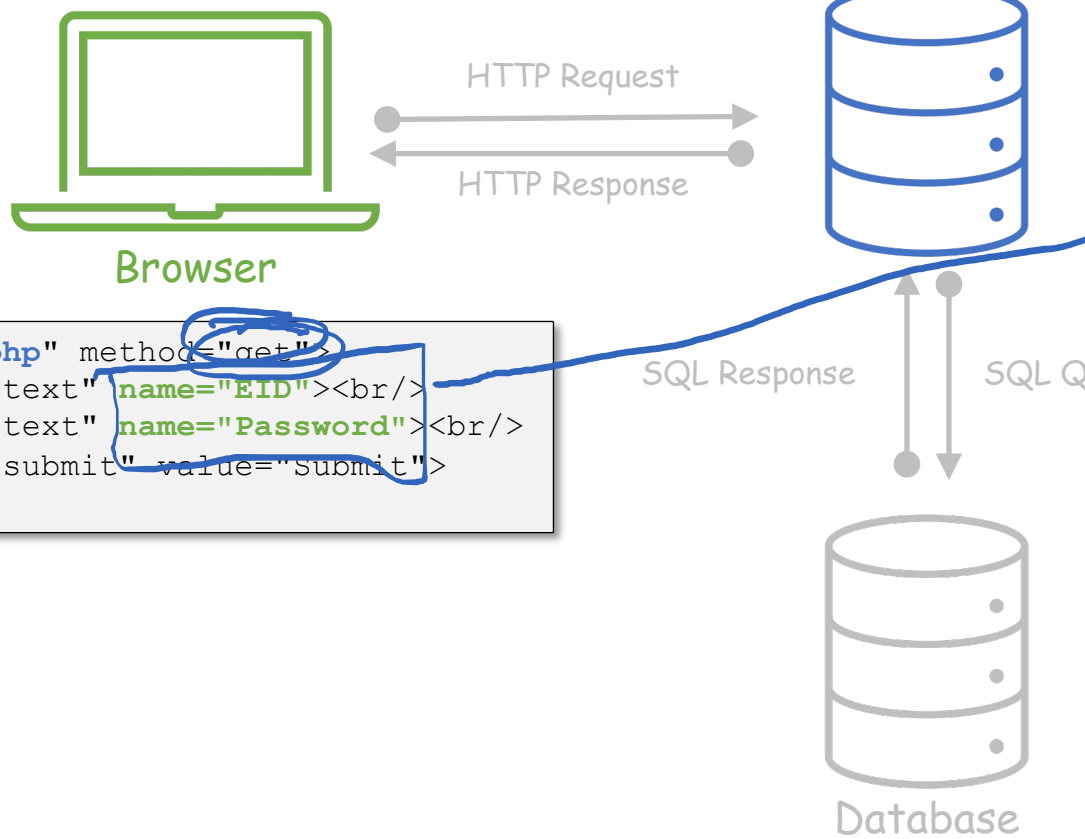
```
http://www.example.com/getdata.php?EID=EID5000&Password=paswd123
```




```
<form action="getdata.php" method="get">
EID:      <input type="text" name="EID"><br/>
Password: <input type="text" name="Password"><br/>
          <input type="submit" value="Submit">
</form>
```

Q: How to get data from a user?

```
http://www.example.com/getdata.php?EID=EID5000&Password=paswd123
```




```php
<?php
    // get data from user request
    $eid = $_GET['EID'];
    $pwd = $_GET['Password'];
    // use params to generate dyn. content
?>
```

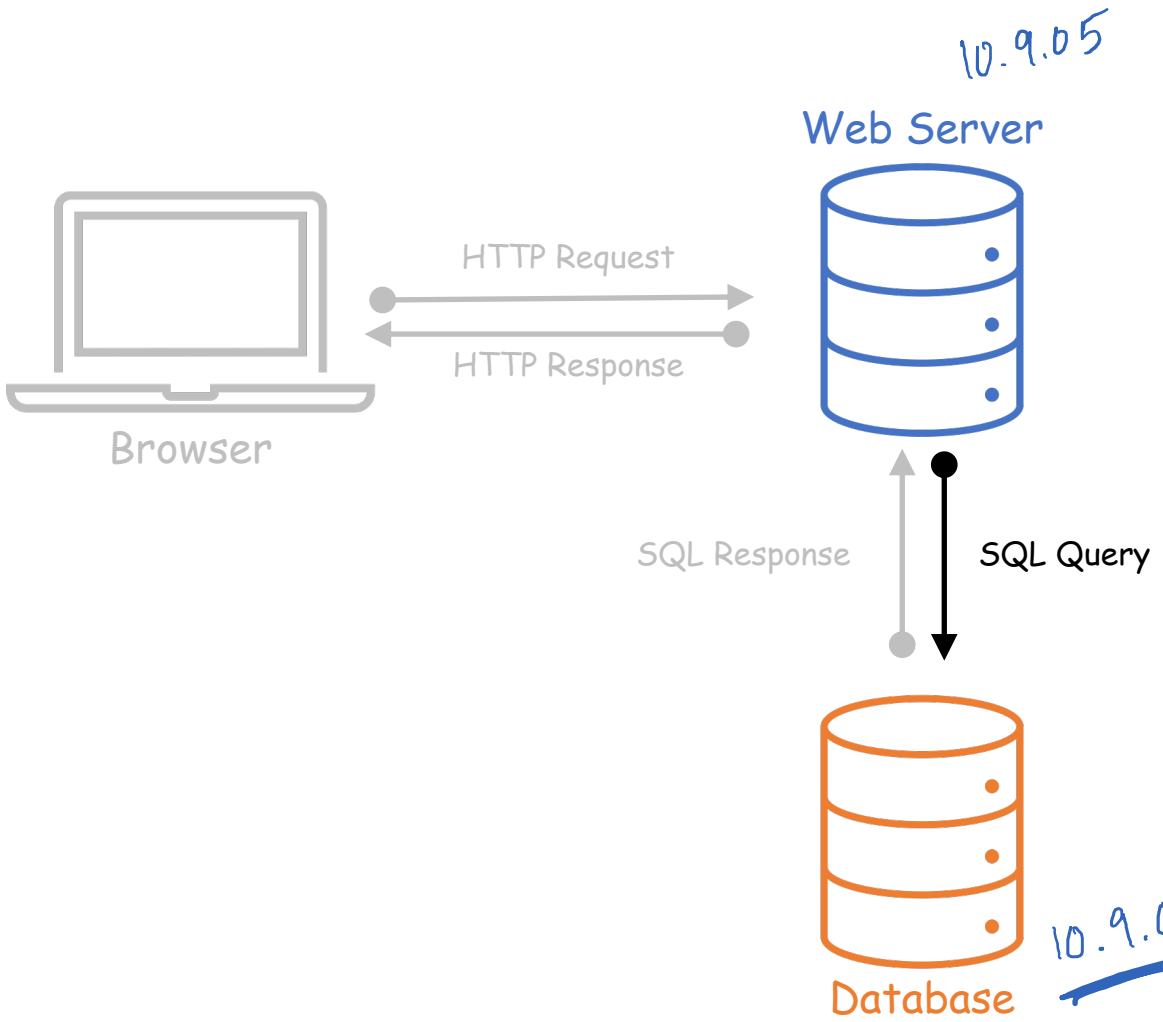
```
<form action="getdata.php" method="get">
EID:      <input type="text" name="EID"><br/>
Password: <input type="text" name="Password"><br/>
          <input type="submit" value="Submit">
</form>
```

HTTPS

POST
- domain / version / path
- header

* body username password

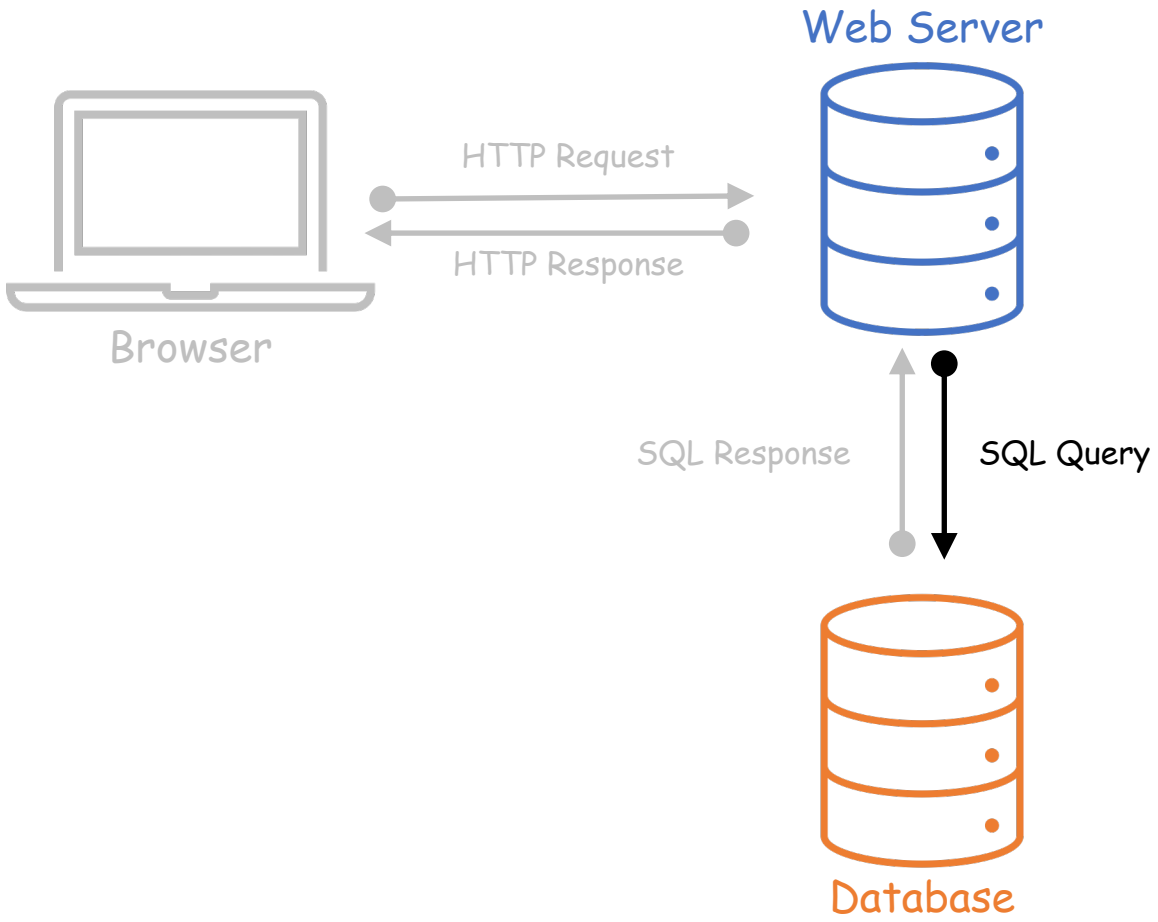
Q: How to get data from a database?




```php
<?php
  ...
  function getDB() {
    $dbhost="10.9.0.6";
    $dbuser="seed";
    $dbpass="dees";
    $dbname="sqllab_users";

    // Create a DB connection
    $conn = new mysqli($dbhost, $dbuser, $dbpass, $dbname);
    if ($conn->connect_error) {
        die("Connection failed: " . $conn->connect_error . "\n");
    }
    return $conn;
  }

  ...process user params...

  // connect to DB & execute query...
  $conn = getDB();
  $sql = "...query string..."
  $result = $conn->query($sql);

  ...process result...
?>
```

format HTML to return to the client

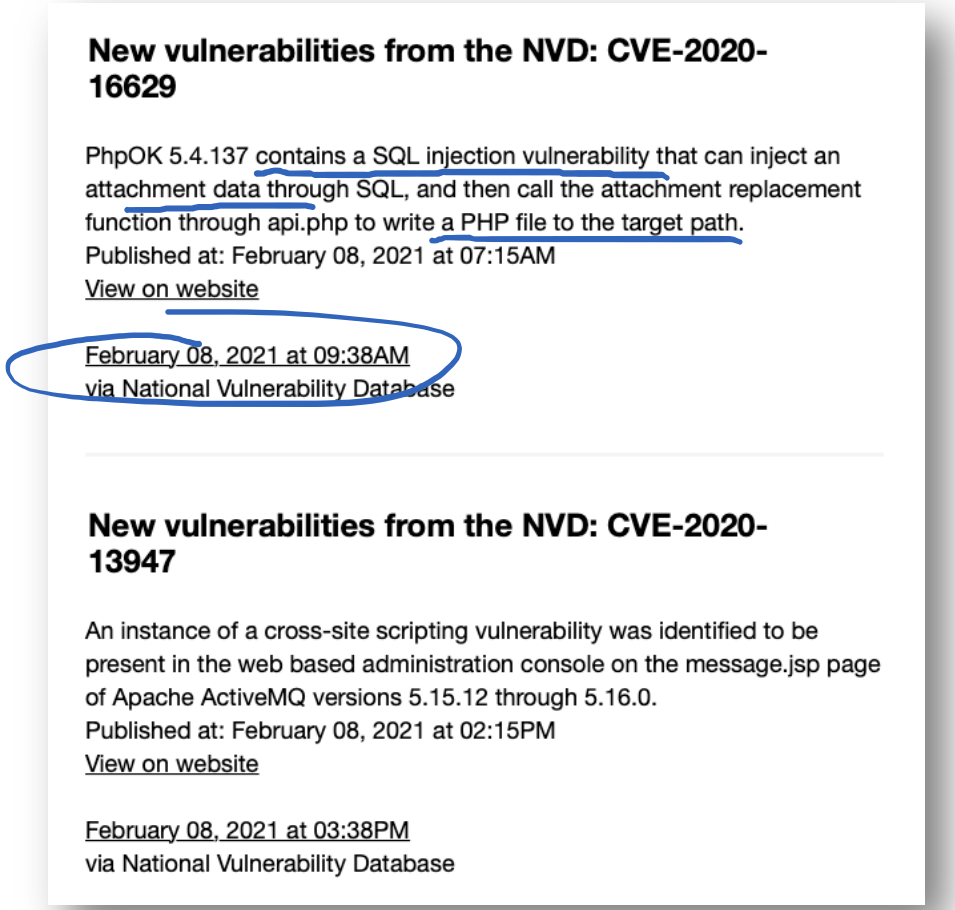
Q: How to get data from a database?




```php
<?php
    ...
    function getDB() {
        $dbhost="10.9.0.6";
        $dbuser="seed";
        $dbpass="dees";
        $dbname="sqllab_users";

        // Create a DB connection
        $conn = new mysqli($dbhost, $dbuser, $dbpass, $dbname);
        if ($conn->connect_error) {
            die("Connection failed: " . $conn->connect_error . "\n");
        }
        return $conn;
    }

    ...process user params...
    $eid = $_GET['EID'];
    $pwd = $_GET['Password'];
    .

    // connect to DB & execute query...
    $conn = getDB();
    $sql = "...query string - includes user params $eid / $pwd!..."
    $result = $conn->query($sql);

    ...process result...

?>
```

SQL Injection

A software vulnerability that occurs when user-supplied data is used as part of a SQL query



New vulnerabilities from the NVD: CVE-2020-16629

PhpOK 5.4.137 contains a SQL injection vulnerability that can inject an attachment data through SQL, and then call the attachment replacement function through api.php to write a PHP file to the target path.
Published at: February 08, 2021 at 07:15AM
View on website

February 08, 2021 at 09:38AM
via National Vulnerability Database

New vulnerabilities from the NVD: CVE-2020-13947

An instance of a cross-site scripting vulnerability was identified to be present in the web based administration console on the message.jsp page of Apache ActiveMQ versions 5.15.12 through 5.16.0.
Published at: February 08, 2021 at 02:15PM
View on website

February 08, 2021 at 03:38PM
via National Vulnerability Database

Launching SQL Injection Attacks

Viewing the Database

- Data provided by the user will become part of the SQL statement.
  Q: Is it possible for a user to change the meaning of the SQL statement?

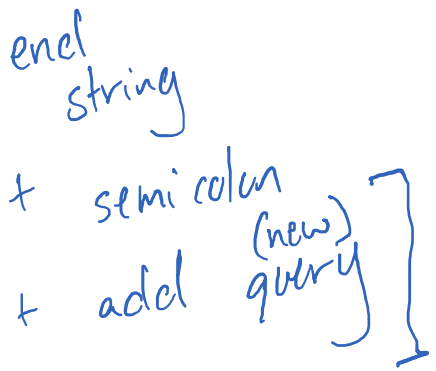
Launching SQL Injection Attacks

Viewing the Database

- Data provided by the user will become part of the SQL statement.
  Q: Is it possible for a user to change the meaning of the SQL statement?

- Web app code enables the user to provide data to fill in details of a query

```
SELECT Name, Salary, SSN
FROM employee
WHERE eid='__________' and password='__________'
```



Launching SQL Injection Attacks

Viewing the Database

- Data provided by the user will become part of the SQL statement.
  Q: Is it possible for a user to change the meaning of the SQL statement?

- Suppose user inputs "**EID5002'#**" in the eid field & "**mypass**" in the password field.
  The SQL statement becomes:

comment

```
SELECT Name, Salary, SSN
FROM employee
WHERE eid=' EID5002'# ' and password= '          '
```

Launching SQL Injection Attacks

Viewing the Database

Q: Could the user enter a string that would be even worse?!

```
SELECT Name, Salary, SSN
FROM employee
WHERE eid='meh' OR 1=1#' and password='_____'
```

OR 1=1

Everyone!

Launching SQL Injection Attacks

Modifying the Database

Q: Could SQL Injection attacks have more severe consequences? E.g., what if the query isn't a SELECT statement, but an UPDATE/INSERT statement?

```
UPDATE emplyee
SET password='_____newpassword_____'
WHERE eid='_____nick eid *_____' and password='_____oldpassword_____';
```

↳ update your salary

UPDATE employee
SET salary='____';

Launching SQL Injection Attacks

Using `curl`

- We can also use command line tools to launch attacks!

```
$ curl 'http://www.example.com/getdata.php?EID=meh' OR 1=1 #&Password='
```

URL escape

- Issues?

Launching SQL Injection Attacks

Using `curl`

- We can also use command line tools to launch attacks!

```
$ curl 'http://www.example.com/getdata.php?EID=meh%27%20OR%201=1%20%23&Password='
```

|----- after encoding -----|

- Pros? Cons?

Launching SQL Injection Attacks

Other Ideas?

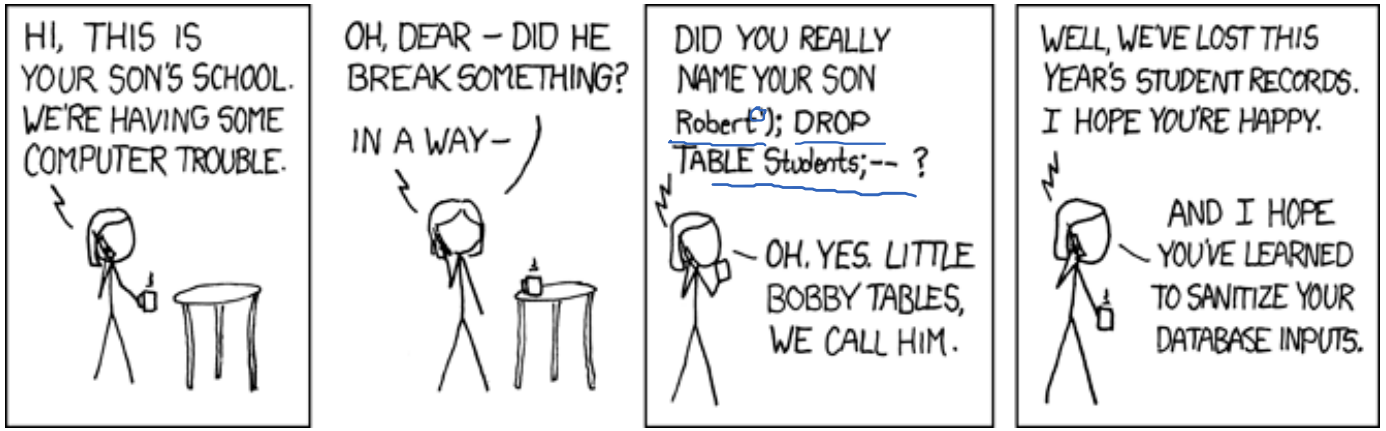
- What else could we potentially do?
  - Execute arbitrary queries?
  - Execute multiple queries?
  - How might you go about doing these things?

SQL Injection Summary

- To bypass the query you need:
  - An always true condition
  - A way to correctly terminate the query / ignore parts of the query

- Can target various SQL queries:
  SELECT, UPDATE, INSERT, etc.

- The fundamental issue (again!)
  -> mixing **code** and **data**!



https://www.xkcd.com/327/

SQL Injection Countermeasures

Filtering & Encoding Data

- Before mixing user-provided data with code, inspect the data and **filter**/**encode** any character that may be interpreted as code.

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

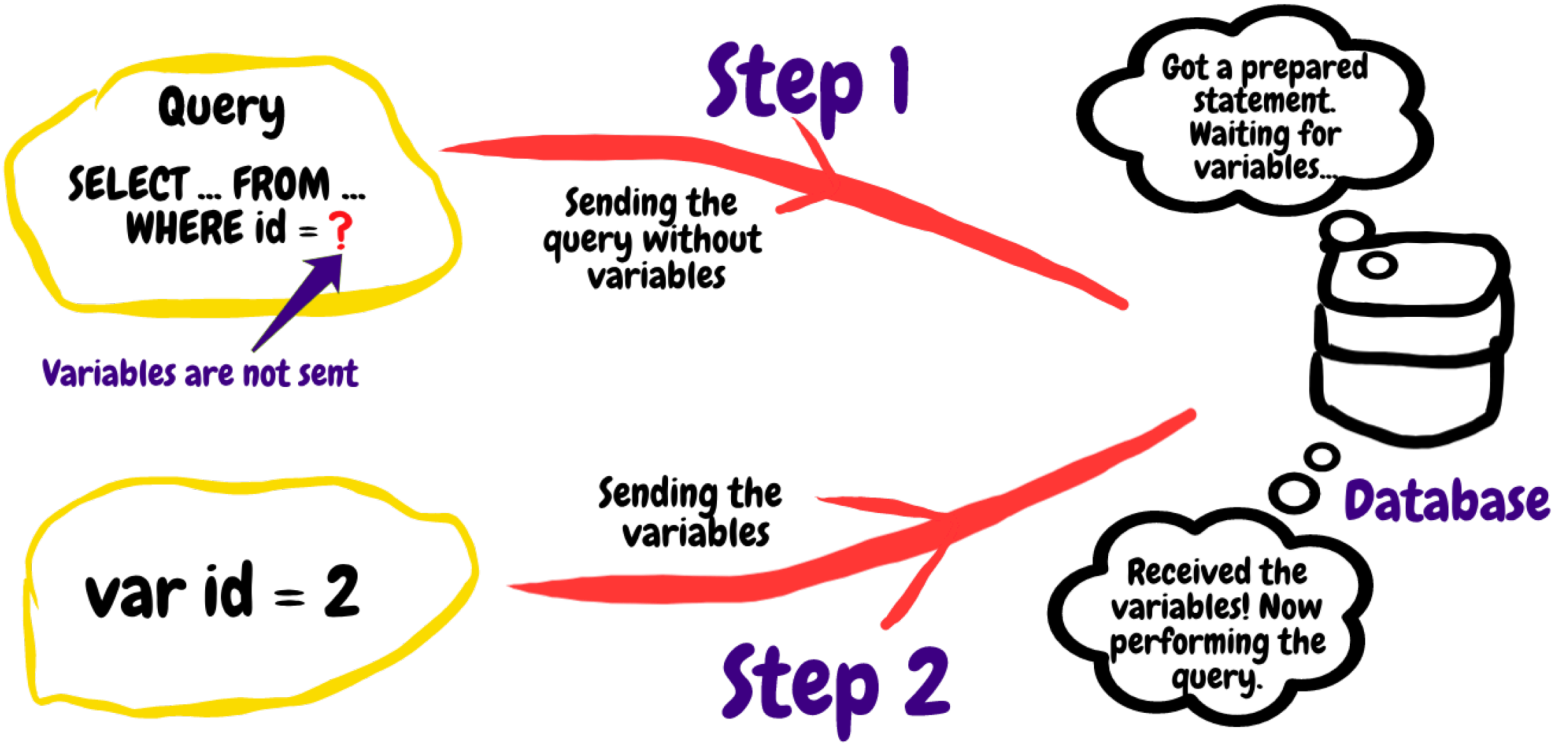
- Most languages have built-in methods or 3rd party extensions to encode/escape characters that have special meaning in the target language. E.g.,
  - real_escape_string
  - htmLawed
  - htmlspecialchars

```
<?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'";
?>
```

- Pros? Cons?

Prepared Statements

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



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

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

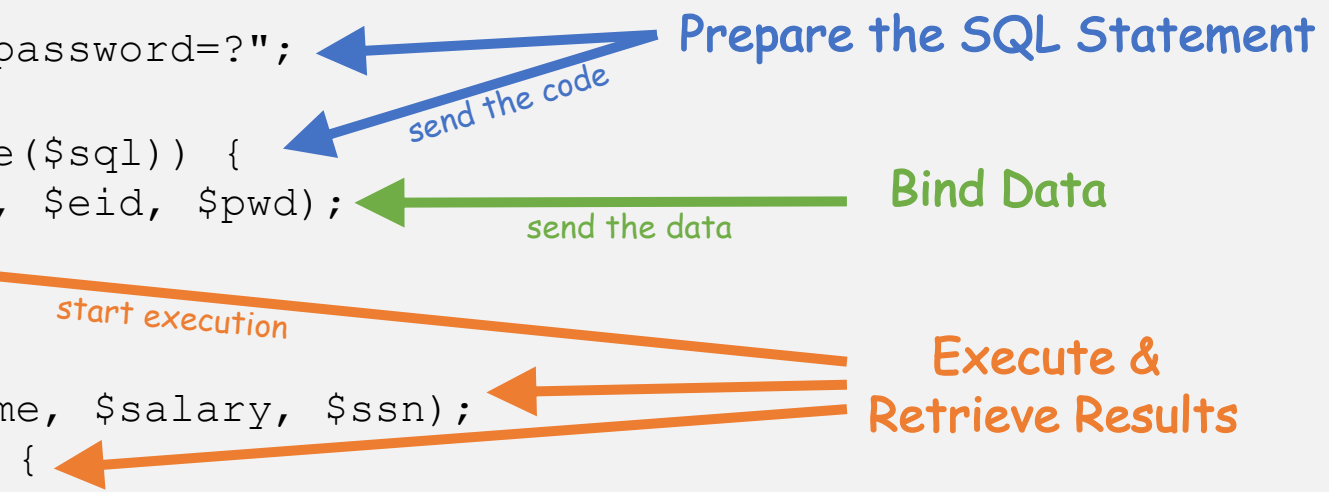
send the code

Bind Data

send the data

Prepared Statements (cont.)

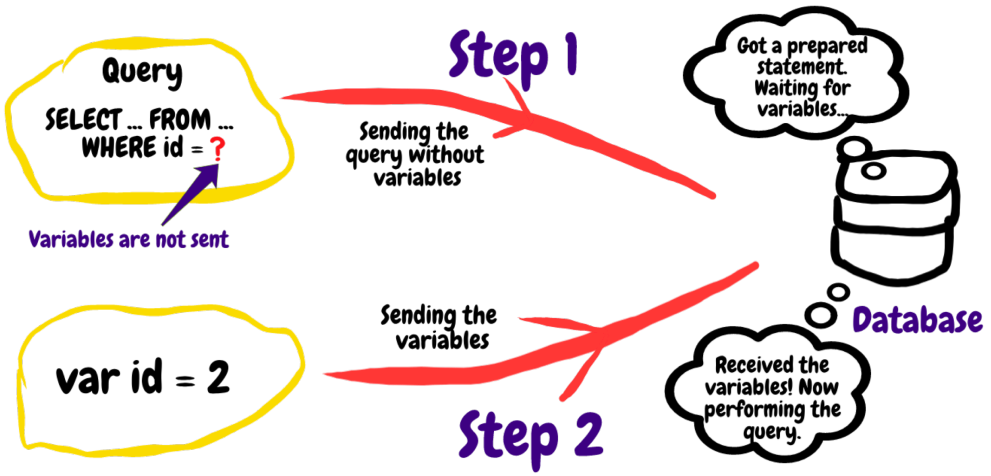
```
/* getdata_prepare.php */
<?php
   $conn = new mysqli("localhost", "root", "seedubuntu", "dbtest");
   $sql = "SELECT Name, Salary, SSN
           FROM employee
           WHERE eid= ? and password=?";

   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);
      }
   }
?>
```



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 / interpretted**
- Attacker can hide code in data, but the code will never be treated as code



Summary

- SQL basics (SELECT, INSERT, UPDATE, etc.)
- The SQL Injection attack w/ examples
- The fundamental cause of SQL Injection vulnerabilities
- How to defend against SQL Injection attacks
  - Data filtering & encoding
  - Prepared statements

You Try!

- 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.