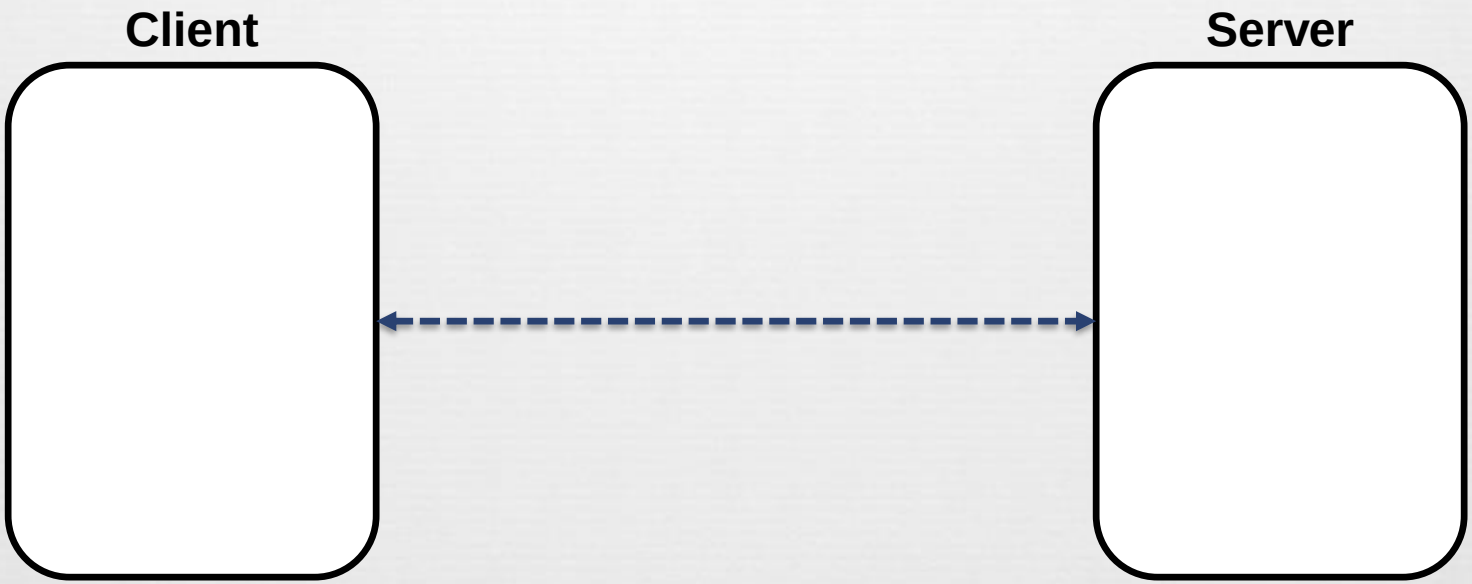
Software Security Assessment

Lab 2: SQL Injection

SQL Injection

The Web, Basically



The Web, Basically

The Web, Basically

The Web, Basically

DB is a separate entity,
Logically (and often physically)

The Web, Basically

(Much) user data is part of the browser

DB is a separate entity, Logically (and often physically)

Server Side Data

Server Side Data

Long-lived state, stored in a separate database

Need to **protect this state** from illicit access and tampering

Server-side data

- Typically want **ACID** transactions

Server-side data

- Typically want **ACID** transactions
  - **A**tomicity
    - Transactions complete entirely or not at all

Server-side data

- Typically want **ACID** transactions
  - **A**tomicity
    - Transactions complete entirely or not at all
  - **C**onsistency
    - The database is always in a valid state

Server-side data

- Typically want **ACID** transactions
  - **A**tomicity
    - Transactions complete entirely or not at all
  - **C**onsistency
    - The database is always in a valid state
  - **I**solation
    - Results from a transaction aren't visible until it is complete

Server-side data

- Typically want **ACID** transactions
  - **A**tomicity
    - Transactions complete entirely or not at all
  - **C**onsistency
    - The database is always in a valid state
  - **I**solation
    - Results from a transaction aren't visible until it is complete
  - **D**urability
    - Once a transaction is committed, its effects persist despite, e.g., power failures

Server-side data

- Typically want **ACID** transactions
  - **A**tomicity
    - Transactions complete entirely or not at all
  - **C**onsistency
    - The database is always in a valid state
  - **I**solation
    - Results from a transaction aren't visible until it is complete
  - **D**urability
    - Once a transaction is committed, its effects persist despite, e.g., power failures

- **Database Management Systems** (DBMSes) provide these properties (and then some)

SQL (Standard Query Language)

Users

Name	Gender	Age	Email	Password
Dee	F	22	dee@x.com	J12jasdj
Mac	M	31	mac@x.com	Jsjaisc4
Bob	M	42	bob@x.com	Csfa8d9a
Alice	F	5	alc@x.com	0csa83asd

SQL (Standard Query Language)

Users

Name	Gender	Age	Email	Password
Dee	F	22	dee@x.com	J12jasdj
Mac	M	31	mac@x.com	Jsjaisc4
Bob	M	42	bob@x.com	Csfa8d9a
Alice	F	5	alc@x.com	0csa83asd

Row (Record)

Column

SQL (Standard Query Language)

Users

Name	Gender	Age	Email	Password
Dee	F	22	dee@x.com	J12jasdj
Mac	M	31	mac@x.com	Jsjaisc4
Bob	M	42	bob@x.com	Csfa8d9a
Alice	F	5	alc@x.com	0csa83asd

```
SELECT Age FROM Users WHERE Name = 'Dee';
```

SQL (Standard Query Language)

Users

Name	Gender	Age	Email	Password
Dee	F	22	dee@x.com	J12jasdj
Mac	M	31	mac@x.com	Jsjaisc4
Bob	M	42	bob@x.com	Csfa8d9a
Alice	F	5	alc@x.com	0csa83asd

22

SELECT Age FROM Users WHERE Name = 'Dee';

SQL (Standard Query Language)

Users

Name	Gender	Age	Email	Password
Dee	F	22	dee@x.com	J12jasdj
Mac	M	31	mac@x.com	Jsjaisc4
Bob	M	42	bob@x.com	Csfa8d9a
Alice	F	5	alc@x.com	0csa83asd

SELECT Age FROM Users WHERE Name = 'Dee';

UPDATE Users SET Email='xxx@x.com' WHERE Age= 42

SQL (Standard Query Language)

Users

Name	Gender	Age	Email	Password
Dee	F	22	dee@x.com	J12jasdj
Mac	M	31	mac@x.com	Jsjaisc4
Bob	M	42	xxx@x.com	Csfa8d9a
Alice	F	5	alc@x.com	0csa83asd

SELECT Age FROM Users WHERE Name = 'Dee';

UPDATE Users SET Email='xxx@x.com' WHERE Age = 42

SQL (Standard Query Language)

Users

Name	Gender	Age	Email	Password
Dee	F	22	dee@x.com	J12jasdj
Mac	M	31	mac@x.com	Jsjaisc4
Bob	M	42	bob@x.com	Csfa8d9a
Alice	F	5	alc@x.com	0csa83asd

```
SELECT  Age  FROM  Users  WHERE  Name =  'Dee';

UPDATE Users SET Email='xxx@x.com'  WHERE  Age=  42

INSERT  INTO Users Values('Frank', 'M', 57, …);
```

SQL (Standard Query Language)

Users

Name	Gender	Age	Email	Password
Dee	F	22	dee@x.com	J12jasdj
Mac	M	31	mac@x.com	Jsjaisc4
Bob	M	42	bob@x.com	Csfa8d9a
Alice	F	5	alc@x.com	0csa83asd
Frank	M	57	frk@x.com	i023djasd

SELECT Age FROM Users WHERE Name = 'Dee';

UPDATE Users SET Email='xxx@x.com' WHERE Age= 42

INSERT INTO Users Values('Frank', 'M', 57, …);

SQL (Standard Query Language)

- More Basics about SQL
  - http://www.w3schools.com/sql/

SQL Injection

Users

Name	Gender	Age	Email	Password
Dee	F	22	dee@x.com	J12jasdj
Mac	M	31	mac@x.com	Jsjaisc4
Bob	M	42	bob@x.com	Csfa8d9a
Alice	F	5	alc@x.com	0csa83asd

```
$result = mysql_query( SELECT  * FROM  Users  WHERE  Name =  '$user');
```

SQL Injection

Users

Name	Gender	Age	Email	Password
Dee	F	22	dee@x.com	J12jasdj
Mac	M	31	mac@x.com	Jsjaisc4
Bob	M	42	bob@x.com	Csfa8d9a
Alice	F	5	alc@x.com	0csa83asd

```
$result = mysql_query( SELECT  * FROM  Users  WHERE  Name =  '$user');
```

Actual value

How Web Application Interact with Database

Connecting to MySQL Database

- PHP program connects to the database server before conducting query on database using.
- The code shown below uses new mysqli(…) along with its 4 arguments to create the database connection.

```
function getDB() {
   $dbhost="localhost";
   $dbuser="root";
   $dbpass="seedubuntu";
   $dbname="dbtest";

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

How Web Application Interact with Database

- Construct the query string and then send it to the database for execution.
- The channel between user and database creates a new attack surface for the database.

```
/* getdata.php */
<?php
   $eid = $_GET['EID'];
   $pwd = $_GET['Password'];

   $conn = new mysqli("localhost", "root", "seedubuntu", "dbtest");
   $sql = "SELECT Name, Salary, SSN
           FROM employee
           WHERE eid= '$eid' and password='$pwd'";

   $result = $conn->query($sql);
   if ($result) {
      // Print out the result
      while ($row = $result->fetch_assoc()) {
        printf ("Name: %s -- Salary: %s -- SSN: %s\n",
                $row["Name"], $row["Salary"], $row['SSN']);
      }
      $result->free();
   }
   $conn->close();
?>
```

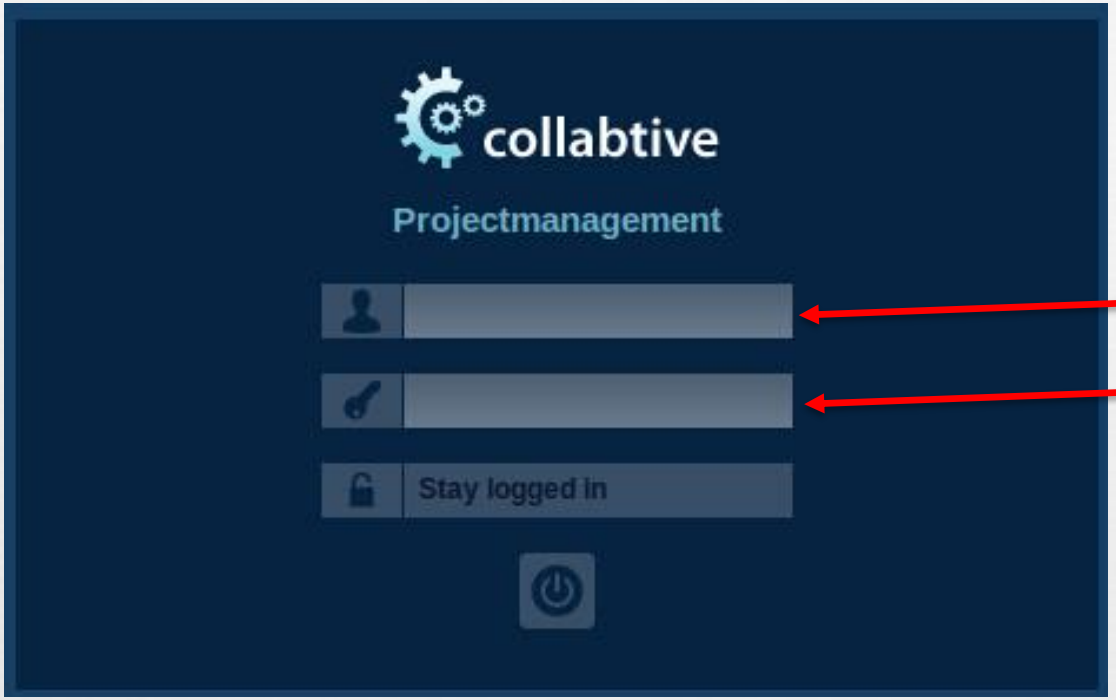
Constructing SQL statement

SQL Injection


```
$result = mysql_query( SELECT  ID, name, local, lastlogin, gender
                       FROM  Users_Table
                       WHERE  name=  '$user' and pass = '$pass');

//If there is at least one record in the database, this user is allowed to login
```

SQL Injection



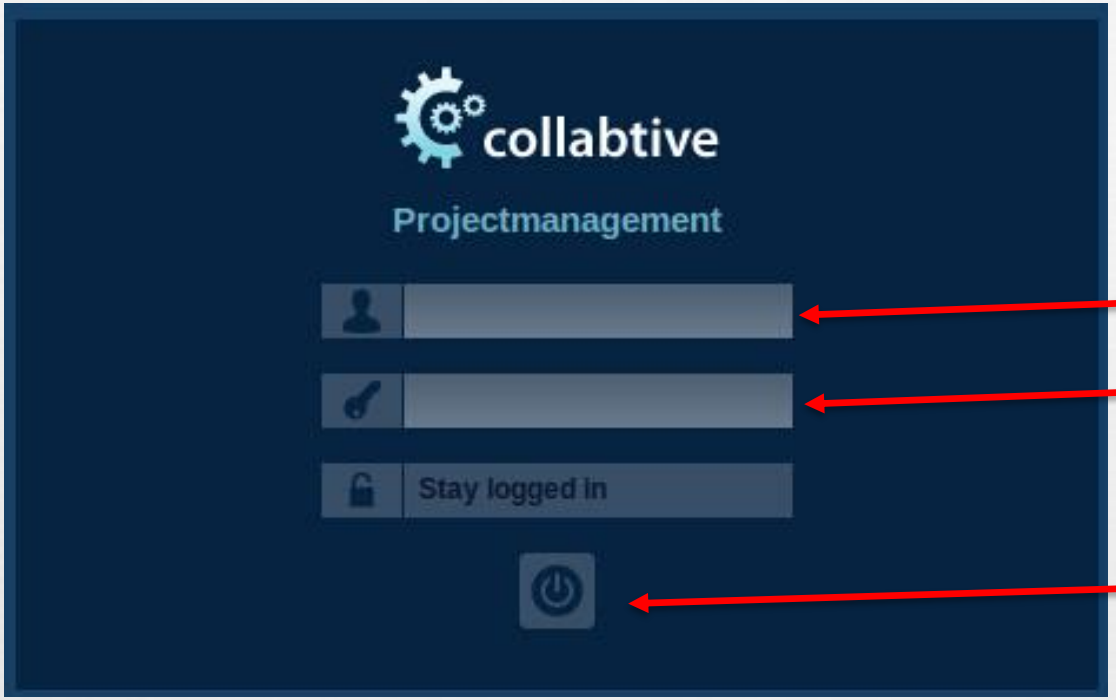
user name

password

```
$result = mysql_query( SELECT  ID, name, local, lastlogin, gender
                       FROM  Users_Table
                       WHERE  name=  '$user' and pass = '$pass');

//If there is at least one record in the database, this user is allowed to login
```

SQL Injection



user name

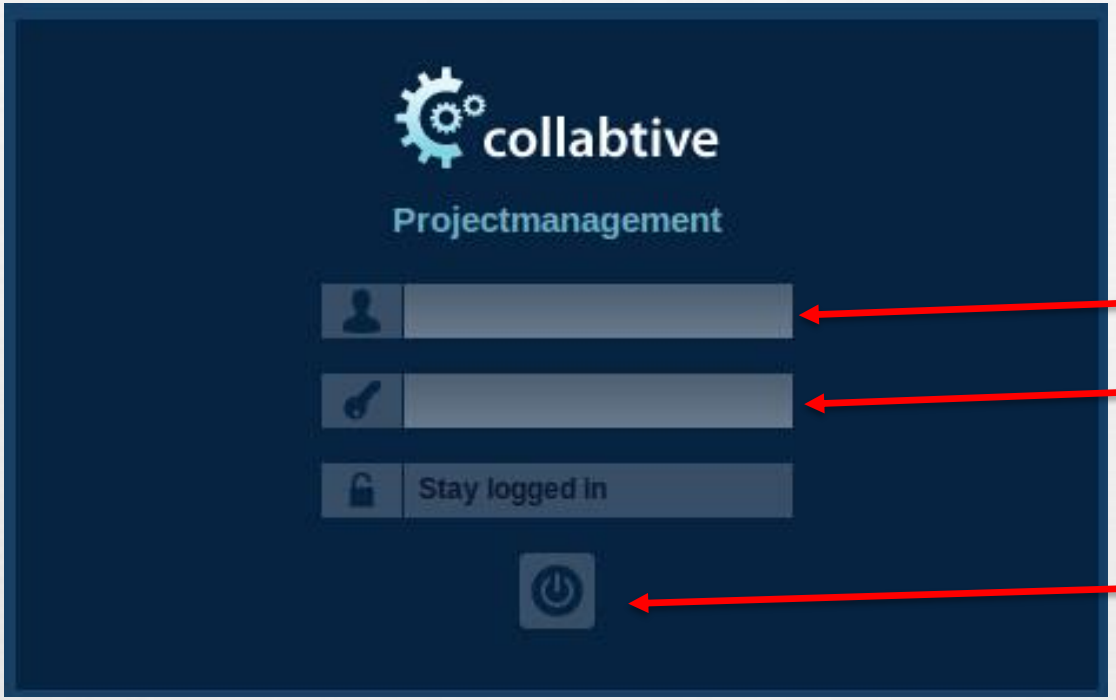
password

HTTP Request Sent

```
$result = mysql_query( SELECT  ID, name, local, lastlogin, gender
                       FROM  Users_Table
                       WHERE  name=  '$user' and pass = '$pass');

//If there is at least one record in the database, this user is allowed to login
```

SQL Injection



user name

password

HTTP Request Sent

```
$result = mysql_query( SELECT  ID, name, local, lastlogin, gender
                       FROM  Users_Table
                       WHERE  name=  'username' and pass = 'password');

//If there is at least one record in the database, this user is allowed to login
```

SQL Injection


```
$result = mysql_query( SELECT  ID, name, local, lastlogin, gender
                       FROM  Users_Table
                       WHERE  name=  '$user' and pass = '$pass');

//If there is at least one record in the database, this user is allowed to login
```

SQL Injection

```
$result = mysql_query( SELECT  ID, name, local, lastlogin, gender
                       FROM  Users_Table

                 WHERE  name= ‘          ’ and pass = ‘                ’;
```

SQL Examples:

```
SELECT * FROM Customers
WHERE City='Berlin' OR City='London';
```

```
SELECT * FROM Customers
WHERE Country='Germany'  AND City='Berlin';
```

SQL Injection

```
$result = mysql_query( SELECT  ID, name, local, lastlogin, gender
                       FROM  Users_Table

                 WHERE  name= ‘          ’ and pass = ‘                ’;
```

In SQL, # is used to denote the comment part

SQL Injection

```
$result = mysql_query( SELECT  ID, name, local, lastlogin, gender
                       FROM  Users_Table

                                                    Skip it
        WHERE  name= '          ' and pass = '              ';
```

In SQL, # is used to denote the comment part

We have no idea about the password, we want to **skip** this checking

SQL Injection

```
$result = mysql_query( SELECT  ID, name, local, lastlogin, gender
                       FROM  Users_Table

                                                  Skip it, use #
        WHERE  name= '          ' and pass = '              ';
```

In SQL, # is used to denote the comment part

We have no idea about the password, we want to **skip** this checking

SQL Injection

```
$result = mysql_query( SELECT  ID, name, local, lastlogin, gender
                       FROM  Users_Table

                 WHERE  name= '        # ' and pass = '              ';
```

Does not work anymore

In SQL, # is used to denote the comment part

We have no idea about the password, we want to **skip** this checking

SQL Injection

```
$result = mysql_query( SELECT  ID, name, local, lastlogin, gender
                       FROM  Users_Table

                 WHERE  name= '        # ' and pass = '              ';
```

In SQL, # is used to denote the comment part

We have no idea about the password, we want to **skip** this checking

Make the rest condition **always to be true**

SQL Injection

```
$result = mysql_query( SELECT  ID, name, local, lastlogin, gender
                       FROM  Users_Table

                       WHERE  name= ‘’ OR 1=1        # ’ and pass = ‘              ’;
```

In SQL, **#** is used to denote the comment part

We have no idea about the password, we want to **skip** this checking

Make the rest condition **always to be true**

User name: **‘ OR 1=1#**
Password: whatever

SQL Injection

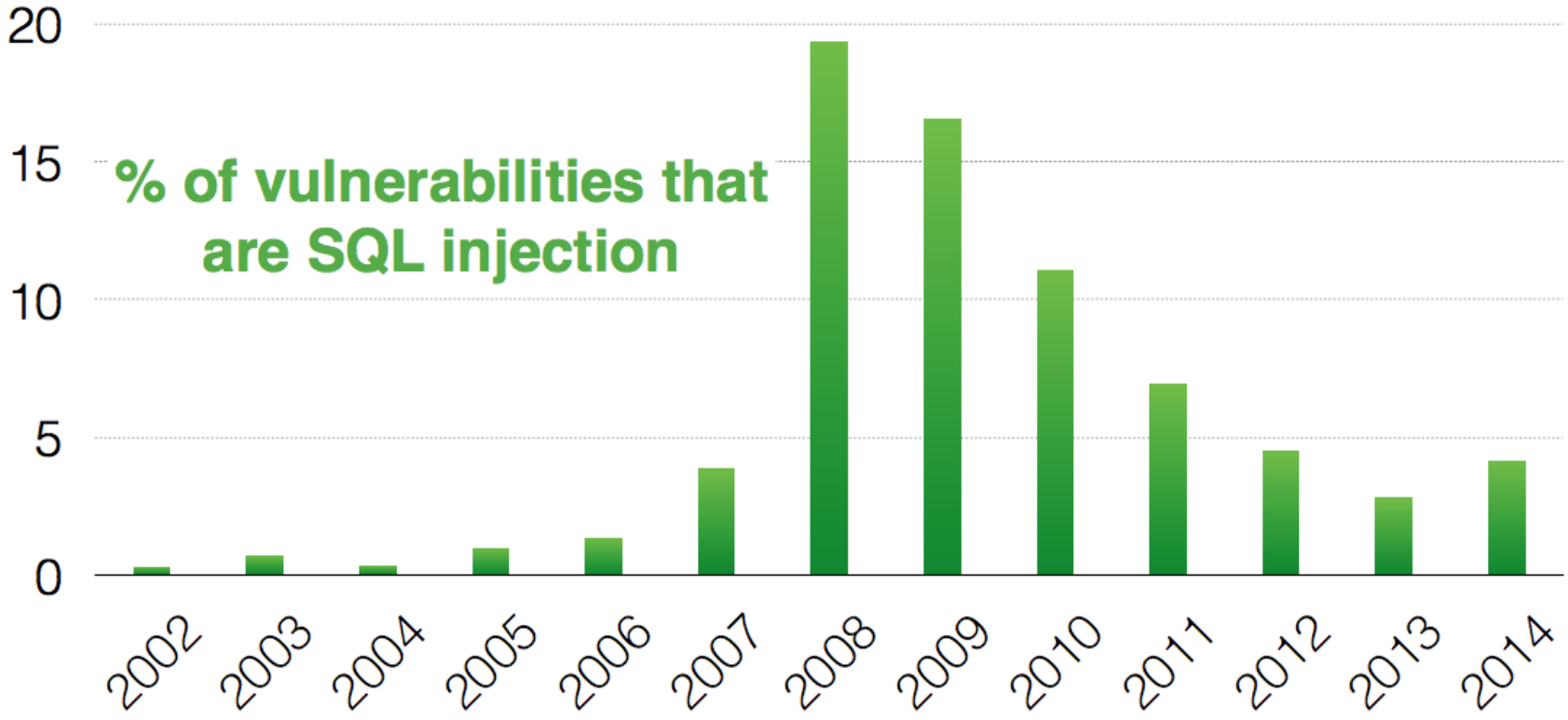
```
$result = mysql_query( SELECT  ID, name, local, lastlogin, gender
                       FROM  Users_Table

                       WHERE  name= ‘’ OR 1=1        #’ and pass = ‘              ’;
```

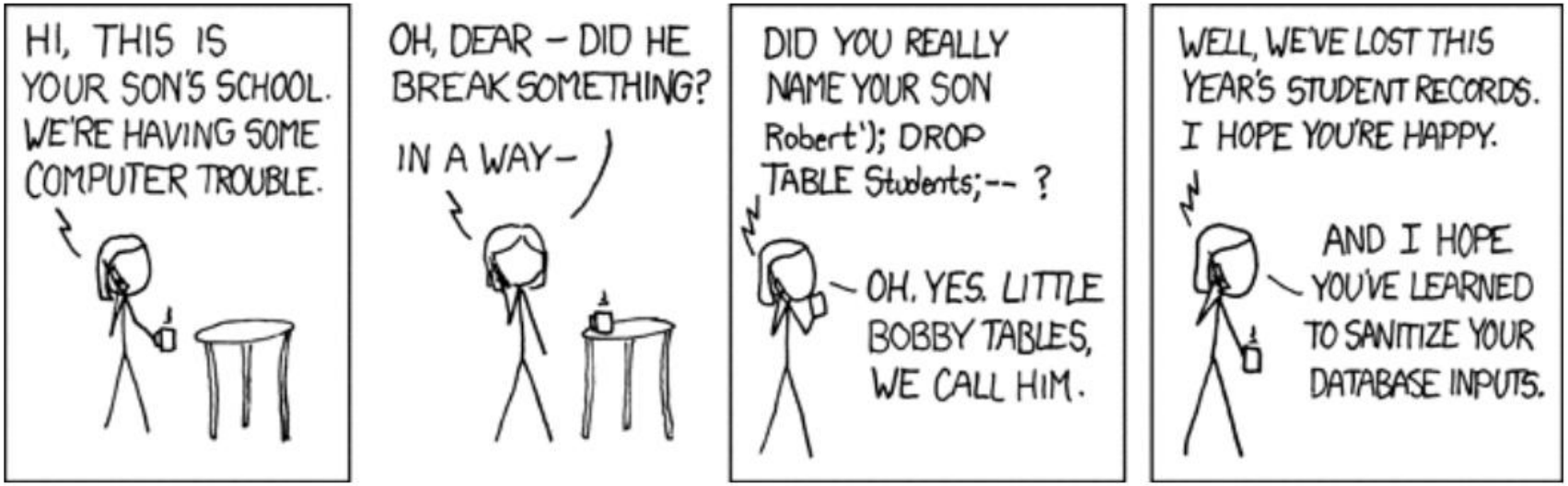
In SQL, # is used to add comment

Can you perform more attacks than login? Like Delete User; Insert new user, etc

- Think about the “;” in our bash, can we find something similar?
- Google UNION operation for SQL



http://web.nvd.nist.gov/view/vuln/statistics



http://xkcd.com/327/


ZU 0666',0,0); DROP DATABASE TABLICE;--

SQL injection countermeasures

The underlying issue

```
$result = mysql_query( SELECT  ID, name, local, lastlogin, gender
                       FROM  Users_Table
                       WHERE  name=  '$user' and pass = '$pass');

//If there is at least one record in the database, this user is allowed to login
```

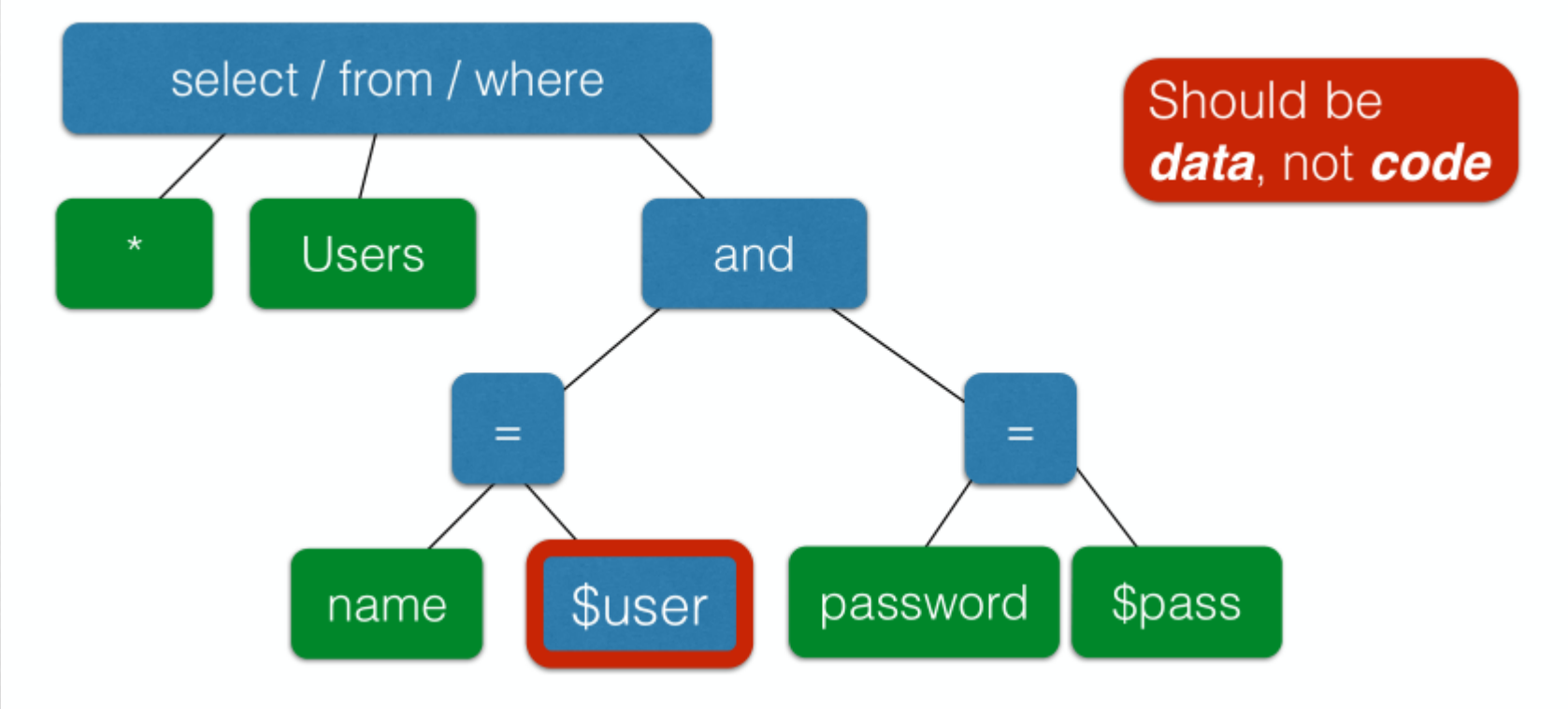
This one string combines the **code** and the **data**

The underlying issue

```
$result = mysql_query( SELECT  ID, name, local, lastlogin, gender
                       FROM  Users_Table
                       WHERE  name=  '$user' and pass = '$pass');

//If there is at least one record in the database, this user is allowed to login
```

This one string combines the **code** and the **data**



Prevention: Input Validation

- Since we require input of a certain form, but we cannot guarantee it has that form, we must validate it before we trust it.

Prevention: Input Validation

- Since we require input of a certain form, but we cannot guarantee it has that form, we must validate it before we trust it.

- **Making input trustworthy!**
- **Check it** has the expected form, and reject it if not

Prevention: Input Validation

- Since we require input of a certain form, but we cannot guarantee it has that form, we must validate it before we trust it.

- **Making input trustworthy!**
- **Check it** has the expected form, and reject it if not
- **Sanitize** it by modifying it or using it in such a way that the result is correctly formed by construction

Sanitization

- Delete the characters you don't want (Blacklist)
  - , ; # -- …
    - **We need these characters, e.g., "Peter O'Connor"**

Sanitization

- Delete the characters you don't want (Blacklist)
  - , ; # -- …
    - **We need these characters, e.g., "Peter O'Connor"**

- **Escaping**

  Replace problematic characters with safe ones!
  - **change ' to \'**
  - **change ; to \;**
  - **change - to \-**
  - **change \ to **

Escaping Special Characters

- "magic_quotes_gpc = On" in php.ini
  - When on, all ' (single-quote), " (double quote), \ (backslash) and *NULL* characters are escaped with a backslash automatically.

Escaping Special Characters

- "magic_quotes_gpc = On" in php.ini
  - When on, all ' (single-quote), " (double quote), \ (backslash) and *NULL* characters are escaped with a backslash automatically.

1. **Any problem for using it?**

Escaping Special Characters

- "magic_quotes_gpc = On" in php.ini
  - When on, all ' (single-quote), " (double quote), \ (backslash) and *NULL* characters are escaped with a backslash automatically.

1. **Any problem for using it?**

   – Portability: Assuming it to be on, or off, affects portability. Most code has to use a function called get_magic quotes_gpc() to check for this, and code accordingly.

   – Performance and Inconvenience: not all user inputs are used for SQL queries, so mandatory escaping all data not only affects performance, but also become annoying when some data are not supposed to be escaped.

   - **For example**, emailing from a form, and seeing a bunch of \' within the email

Escaping Special Characters

- “mysql_real_escape_string()” in php.code
  - Escapes special characters in a string for use in an SQL statement
  - mysql_real_escape_string() calls MySQL's library function mysql_real_escape_string, which prepends backslashes to the following characters: *\x00*, *\n*, *\r*, \, ', " and *\x1a*.

Escaping Special Characters

- “mysql_real_escape_string()” in php.code
  - Escapes special characters in a string for use in an SQL statement
  - mysql_real_escape_string() calls MySQL's library function mysql_real_escape_string, which prepends backslashes to the following characters: *\x00*, *\n*, *\r*, **, *'*, *"* and *\x1a*.

```php
<?php
// Connect
$link = mysql_connect('mysql_host', 'mysql_user', 'mysql_password')
    OR die(mysql_error());

// Query
$query = sprintf("SELECT * FROM users WHERE user='%s' AND password='%s'",
            mysql_real_escape_string($user),
            mysql_real_escape_string($password));
?>
```

Prepared Statements

- **Fundament cause** of SQL injection: mixing data and code
- **Fundament solution**: separate data and code.
- **Main Idea:** Sending code and data in separate channels to the database server. This way the database server knows not to retrieve any code from the data channel.
- **How:** using **prepared statement**
- **Prepared Statement:** It is an optimized feature that provides improved performance if the same or similar SQL statement needs to be executed repeatedly. Using prepared statements, we send an SQL statement template to the database, with certain values called parameters left unspecified. The database parses, compiles and performs query optimization on the SQL statement template and stores the result without executing it. We later bind data to the prepared statement

Prepared Statements

```
$conn = new mysqli("localhost", "root", "seedubuntu", "dbtest");
$sql = "SELECT Name, Salary, SSN
        FROM employee
        WHERE eid= '$eid' and password='$pwd'";
$result = $conn->query($sql);
```

The vulnerable version: code and data are mixed together.

Using prepared statements, we separate code and data.

```
$conn = new mysqli("localhost", "root", "seedubuntu", "dbtest");
$sql = "SELECT Name, Salary, SSN
        FROM employee
        WHERE eid= ? and password=?";                ①

if ($stmt = $conn->prepare($sql)) {                  ②
   $stmt->bind_param("ss", $eid, $pwd);              ③
   $stmt->execute();                                 ④

   $stmt->bind_result($name, $salary, $ssn);         ⑤
   while ($stmt->fetch()) {                          ⑥
      printf ("%s %s %s\n", $name, $salary, $ssn);
   }
}
```

Send code

Send data

Start execution

Prepared Statements

- 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, so it will never be attacked.