Web Security

Lecture 18

Instructor: Dr. Cong Pu, Ph.D.

cong.pu@okstate.edu

Acknowledgment: Adapted partially from course materials from Dr. Wenliang Du at Syracuse University, Dr. Fengwei Zhang at Southern University of Science and Technology, and Dr. Steven M. Bellovin at Columbia University.

Introduction

- Real-world web app., data are stored in *database*
  - *save* data to or *get* data from database
    - construct *SQL* statement
    - send *SQL* statement to *database*
      - *database*
        - execute *SQL* statement
        - return results back to web app.
- SQL statement *usually* contains ***user-provide data***
  - what if a SQL statement is not constructed properly?
    - injecting ***code*** into *SQL* statement
      - cause database to execute ***code***
      - SQL injection vulnerability

A Brief Tutorial of SQL

- Login to database, e.g., MySQL:
  - use MySQL database, which is an open-source relational database management system
  - login using the following command

```
$ mysql -uroot -pseedubuntu
Welcome to the MySQL monitor.
...
mysql>
```

login name password

mysql prompt: indicating login successfully

Note:
- space between –u and login name
- space between –p and password

- create database:
  - inside MySQL, create multiple databases
    - *'SHOW DATABSES'* command can be used to list existing databases
  - create new database called *dbtest*

```
mysql> SHOW DATABASES;
......
mysql> CREATE DATABASE dbtest;
```

create database command

SQL commands are not case sensitive
- using upper-case to separate from non-commands in lower-case

SQL Tutorial: Create a Table

- Relational database organizes data using tables
  - database has multiple tables
- create table called *employee* with seven attributes (i.e., columns) for the database *dbtest*

select database to use

```
mysql> USE dbtest
mysql> CREATE TABLE employee (
  ID       INT (6) NOT NULL AUTO_INCREMENT,
  Name     VARCHAR (30) NOT NULL,
  EID      VARCHAR (7) NOT NULL,
  Password VARCHAR (60),
  Salary   INT (10),
  SSN      VARCHAR (11),
  PRIMARY KEY (ID)
);
mysql> DESCRIBE employee;
+----------+-------------+------+-----+---------+----------------+
| Field    | Type        | Null | Key | Default | Extra          |
+----------+-------------+------+-----+---------+----------------+
| ID       | int(6)      | NO   | PRI | NULL    | auto_increment |
| Name     | varchar(30) | NO   |     | NULL    |                |
| EID      | varchar(30) | NO   |     | NULL    |                |
| Password | varchar(60) | YES  |     | NULL    |                |
| Salary   | int(10)     | YES  |     | NULL    |                |
| SSN      | varchar(11) | YES  |     | NULL    |                |
+----------+-------------+------+-----+---------+----------------+
```

define the structure of table 'employee'

- table columns are defined inside parentheses
  - each column contains
    - name, followed by type
    - number: maximum length
    - constraints (i.e., NOT NULL)

display the structure of table 'employee'

SQL Tutorial: Insert a Row

- use the '*INSERT INTO*' statement to insert new record into table:

```
mysql> INSERT INTO employee (Name, EID, Password, Salary, SSN)
       VALUES ('Ryan Smith', 'EID5000', 'paswd123', 80000,
               '555-55-5555');
```

  - insert a record into *employee* table
  - did not specify a value of the *ID column*, as it will be automatically set by the database

SQL Tutorial: Insert a Row

- the '*SELECT*' statement is the most common operation on databases
  - retrieves information from database

all records

```
mysql> SELECT * FROM employee;
+----+---------+---------+----------+--------+-------------+
| ID | Name    | EID     | Password | Salary | SSN         |
+----+---------+---------+----------+--------+-------------+
|  1 | Alice   | EID5000 | paswd123 |  80000 | 555-55-5555 |
|  2 | Bob     | EID5001 | paswd123 |  80000 | 555-66-5555 |
|  3 | Charlie | EID5002 | paswd123 |  80000 | 555-77-5555 |
|  4 | David   | EID5003 | paswd123 |  80000 | 555-88-5555 |
+----+---------+---------+----------+--------+-------------+
mysql> SELECT Name, EID, Salary FROM employee;
+---------+---------+--------+
| Name    | EID     | Salary |
+---------+---------+--------+
| Alice   | EID5000 |  80000 |
| Bob     | EID5001 |  80000 |
| Charlie | EID5002 |  80000 |
| David   | EID5003 |  80000 |
+---------+---------+--------+
```

asks the database for all its records, including all the columns

asks the database only for Name, EID and Salary columns

SQL Tutorial: WHERE Clause

- it is uncommon for a SQL query to retrieve all records in database
- '*WHERE*' clause is used to set conditions for several types of SQL statements including '*SELECT*', '*UPDATE*', '*DELETE*', etc.

```
mysql> SQL Statement
       WHERE predicate;
```

  - the above SQL statement only affects the rows for which the predicate in the '*WHERE*' clause is *TRUE*
    - row for which predicate evaluates to *FALSE* or *Unknown* are *not affected*
  - the predicate is a logical expression
    - multiple predicates can be combined using keywords *AND* and *OR*

SQL Tutorial: WHERE Clause

```
mysql> SELECT * FROM employee WHERE EID='EID5001';
+----+------+---------+----------+--------+-------------+
| ID | Name | EID     | Password | Salary | SSN         |
+----+------+---------+----------+--------+-------------+
|  2 | Bob  | EID5001 | paswd123 |  80000 | 555-66-5555 |
+----+------+---------+----------+--------+-------------+

mysql> SELECT * FROM employee WHERE EID='EID5001' OR Name='David';
+----+-------+---------+----------+--------+-------------+
| ID | Name  | EID     | Password | Salary | SSN         |
+----+-------+---------+----------+--------+-------------+
|  2 | Bob   | EID5001 | paswd123 |  80000 | 555-66-5555 |
|  4 | David | EID5003 | paswd123 |  80000 | 555-88-5555 |
+----+-------+---------+----------+--------+-------------+
```

- 1st query: return a record that has EID5001 in the EID field
- 2nd query: return the records that satisfy either EID = 'EID5001' or Name = 'David'

SQL Tutorial: WHERE Clause

- if the condition is always *True*, then all the rows are affected by SQL statement

```
mysql> SELECT * FROM employee WHERE 1=1;
+----+---------+---------+----------+--------+-------------+
| ID | Name    | EID     | Password | Salary | SSN         |
+----+---------+---------+----------+--------+-------------+
|  1 | Alice   | EID5000 | paswd123 |  80000 | 555-55-5555 |
|  2 | Bob     | EID5001 | paswd123 |  80000 | 555-66-5555 |
|  3 | Charlie | EID5002 | paswd123 |  80000 | 555-77-5555 |
|  4 | David   | EID5003 | paswd123 |  80000 | 555-88-5555 |
+----+---------+---------+----------+--------+-------------+
```

- this **1 = 1** predicate looks quite useless in real queries
  - useful in SQL Injection attacks

SQL Tutorial: UPDATE Statement

- use the '*UPDATE*' Statement to modify an existing record

multiple columns separated by comma

```
mysql> UPDATE employee SET Salary=82000 WHERE Name='Bob';
mysql> SELECT * FROM employee WHERE Name='Bob';
+----+------+---------+----------+--------+-------------+
| ID | Name | EID     | Password | Salary | SSN         |
+----+------+---------+----------+--------+-------------+
|  2 | Bob  | EID5001 | paswd123 |  82000 | 555-66-5555 |
+----+------+---------+----------+--------+-------------+
```

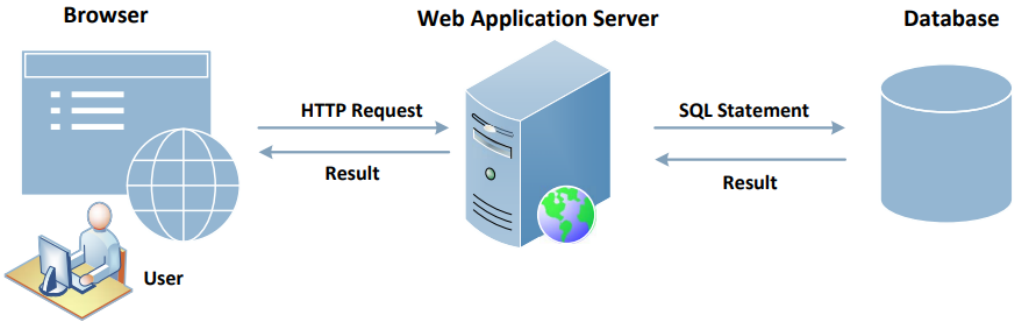
SQL Tutorial: Comments

- MySQL supports three comment styles
  - text from the # character to the end of line is treated as a comment
  - text from the -- _ (space) to the end of line is treated as a comment
    - this style requires the second dash to be followed by at least one whitespace or control character
  - similar to C language, text between /* and */ is treated as a comment
    - this style allows comment to be inserted into the middle of SQL statement; commend can span multiple lines

```
mysql> SELECT * FROM employee;    # Comment to the end of line
mysql> SELECT * FROM employee;    -- Comment to the end of line
mysql> SELECT * FROM /* In-line comment */ employee;
```

Interacting with Database in Web Application

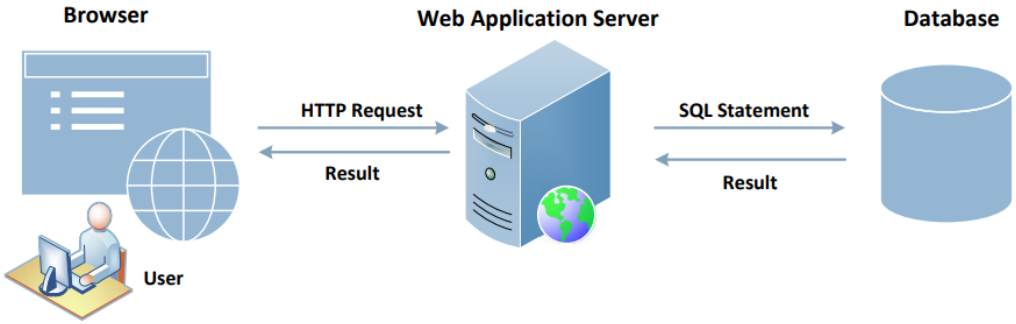
- Typical web app. consists of three major components:



  - web browser
    - get content; present content; interact with user; get user input
    - communicate with web app. server using HTTP or HTTPS
  - web app. server
    - generate and deliver content to browser; rely on independent database server for data management
    - interact with database using SQL
  - database

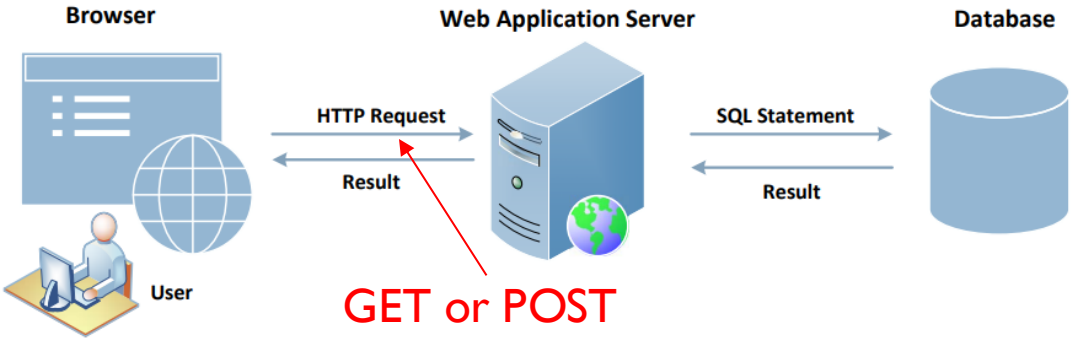
Interacting with Database in Web Application

- Typical web app. consists of three major components:



- ***SQL injection attacks*** can cause ***damage*** to database
- users do not directly interact with database but through web server
  - web app. server provide a channel for user's data to reach database
  - if this channel is not implemented properly, malicious users can attack database

Getting Data from User



- Form where users can type their data
  - once '*Submit*' button is clicked, an HTTP request will be sent out with data attached

EID EID5000
Password paswd123
Submit

- HTML source of the above form is given below:

```
<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>
```

- request generated is:

name of input field

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

Getting Data from User

```
<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>
```

- HTTP GET request
  - method field in HTML code specified GET type
  - in GET requests, parameters are attached after the question mark ? in the URL

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

  - each parameter has a name=value pair and are separated by "&"
  - in the case of HTTPS, the format would be similar but the data will be encrypted
- once this request reached the target PHP script (getdata.php)
  - the parameters inside HTTP request will be saved to an array $_GET or $_POST
  - an example shows a PHP script getting data from GET request

```
<?php
   $eid = $_GET['EID'];
   $pwd = $_GET['Password'];
   echo "EID: $eid --- Password: $pwd\n";
?>
```

$_GET: an associative array of variables passed to the current script via the URL parameters

Launching SQL Injection Attacks

- user input will become part of the SQL statement
  - is it possible for a user to change the meaning of the SQL statement?
- example: the intention of the web app developer by the following is for user to provide some data for the blank areas

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

  - what if user inputs a random string in the password entry and types "EID5002' #" in the eid entry
  - the SQL statement will become the following

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

everything from # sign to the end of line is considered as comment

Launching SQL Injection Attacks

- the SQL statement will be equivalent to the following:

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

  - return the name, salary and SSN of the employee whose EID is EID5002 even though the user doesn't know the employee's password.
- let's see if a user can get all the records from the database
  - assuming that we don't know all the EID's in the database
  - create a predicate for 'WHERE' clause so that it is true for all records

```
SELECT Name, Salary, SSN
FROM employee
WHERE eid= 'a' OR 1=1
```

always true