SQL Injection Attack

CSCE 465

Brief Tutorial of SQL

- **Log in to MySQL:** We will use MySQL database, which is an open-source relational database management system. We can log in using the following command:

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

- **Create a Database**: Inside MySQL, we can create multiple databases. "SHOW DATABSES" command can be used to list existing databases. We will create a new database called dbtest:

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

SQL Tutorial: Create a Table

- A relational database organizes its data using tables. Let us create a table called employee with seven attributes (i.e. columns) for the database "dbtest"

  - We need to let the system know which database to use as there may be multiple databases

  - After a table is created, we can use describe to display the structure of the table

```
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    |                |
+----------+-------------+------+-----+---------+----------------+
```

SQL Tutorial: Insert a Row

- We can use the INSERT INTO statement to insert a new record into a table :

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

- Here, we insert a record into the "employee" table.

- We do not specify a value of the ID column, as it will be automatically set by the database.

SQL Tutorial: SELECT Statement

- The SELECT statement is the most common operation on databases
- It retrieves information from a database

```
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 a 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 reflects the rows for which the predicate in the WHERE clause is TRUE.

- The predicate is a logical expression; multiple predicates can be combined using keywords AND and OR.

- Lets look at an example in the next slide.

SQL Tutorial: WHERE Clause

- The first query returns a record that has EID5001 in EID field
- The second query returns the records that satisfy either EID='EID5001' or Name='David'

```
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 |
+----+-------+---------+----------+--------+-------------+
```

SQL Tutorial: WHERE Clause

- If the condition is always True, then all the rows are affected by the 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, but it will become useful in SQL Injection attacks

SQL Tutorial: UPDATE Statement

- We can use the UPDATE Statement to modify an existing record

```
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 "--" to the end of line is treated as a comment.
- Similar to C language, text between /* and */ is treated as a comment

```
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
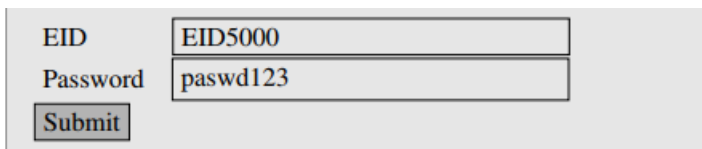
- A typical web application consists of three major components:
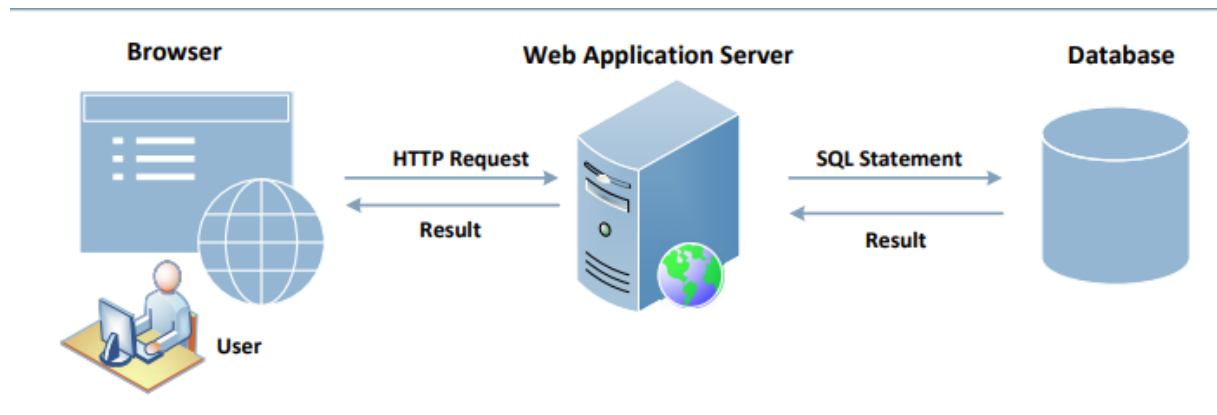
- SQL Injection attacks can cause damage to the database. As we notice in the figure, the users do not directly interact with the database but through a web server. If this channel is not implemented properly, malicious users can attack the database.

Getting Data from User

- This example shows a form where users can type their data. Once the submit button is clicked, an HTTP request will be sent out with the data attached

EID	EID5000
Password	paswd123
Submit	

- The 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:

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

Getting Data from User

- The request shown is an HTTP GET request, because the method field in the HTML code specified the get type

- In GET requests, parameters are attached after the question mark in the URL

- 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 the parameters inside the HTTP request will be saved to an array $_GET or $_POST. The following example shows a PHP script getting data from a GET request

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

How Web Applications 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.

```php
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 Applications 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.

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

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

   $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();
?>
```
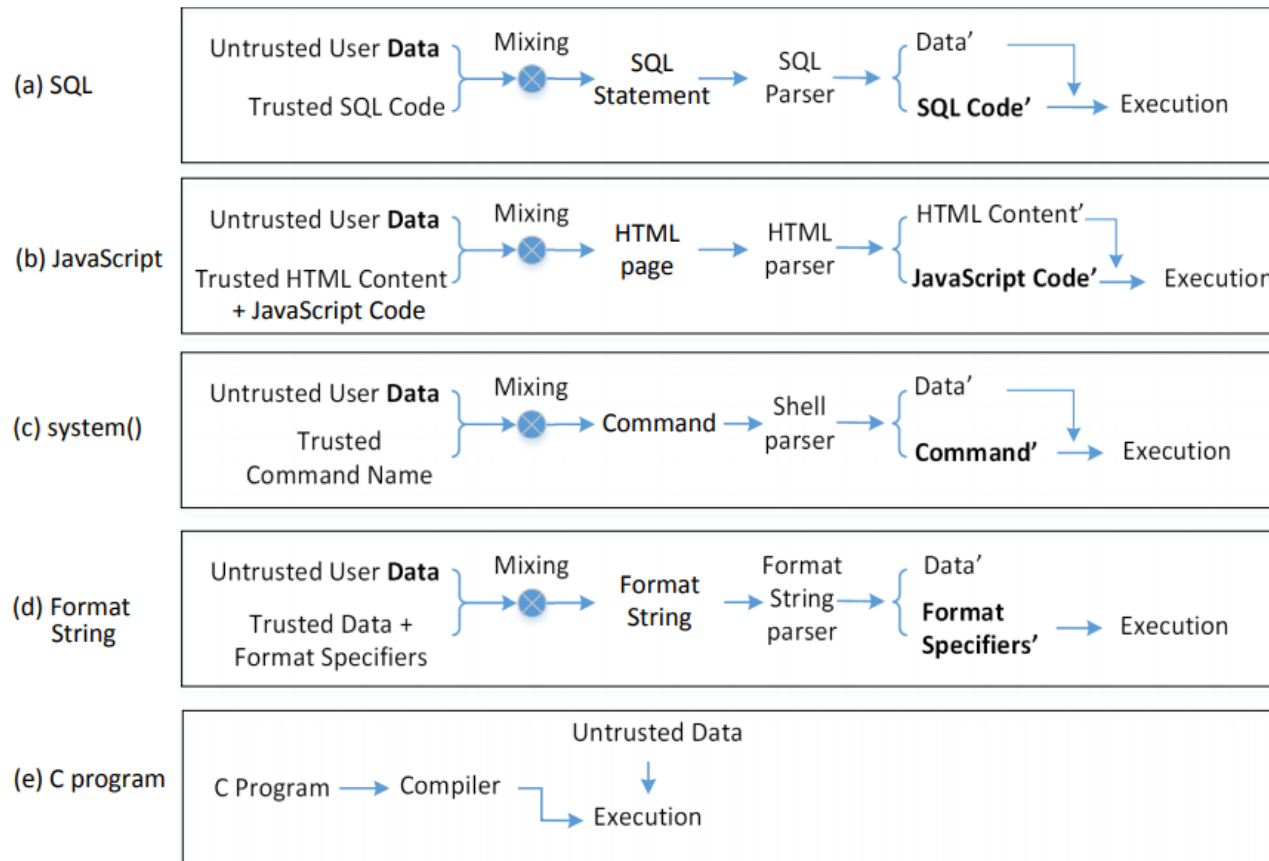
Launching SQL Injection Attacks

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

- The intention of the web app developer by the following is for the user to provide some data for the blank areas.

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

The Fundamental Cause

Mixing data and code together is the cause of several types of vulnerabilities and attacks including SQL Injection attack, XSS attack, attacks on the system() function and format string attacks.

Countermeasures: Filtering and 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. This can be seen in the following example

```
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(). It can be used to encode the characters that have special meanings in SQL. The following code snippet shows how to use this API.

```
/* getdata_encoding.php */
<?php
   $conn = new mysqli("localhost", "root", "seedubuntu", "dbtest");
   $eid = $mysqli->real_escape_string($_GET['EID']);          ①
   $pwd = $mysqli->real_escape_string($_GET['Password'];      ②
   $sql = "SELECT Name, Salary, SSN
             FROM employee
             WHERE eid= '$eid' and password='$pwd'";
?>
```

Countermeasures: Prepared Statement

- **Fundamental cause** of SQL injection: mixing data and code

- **Fundamental 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

Countermeasures: Prepared Statement

```
$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 execusion

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, so it will never be attacked.

Summary

- Brief tutorial of SQL
- SQL Injection attack and how to launch this type of attacks
- The fundament cause of the vulnerability?
- How to defend against SQL Injection attacks?
- Prepared Statement