Web Security

Ethical Hacking

Alessandro Brighente
Eleonora Losiouk

Master Degree on Cybersecurity

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

SPRITZ
SECURITY & PRIVACY
RESEARCH GROUP

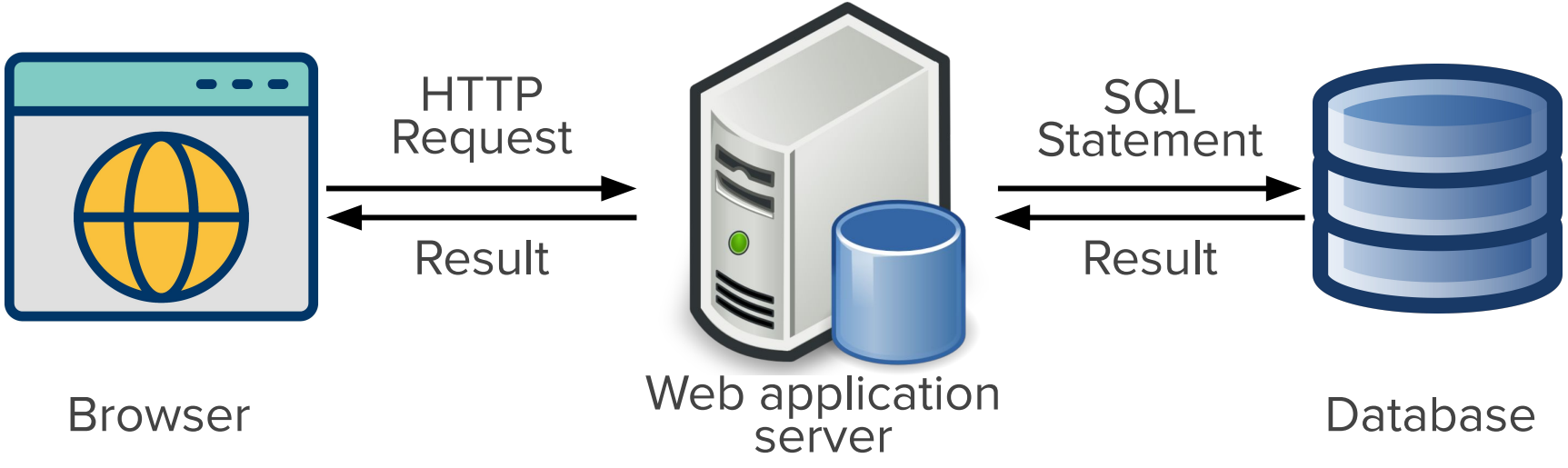
Outline

SPRITZ
SECURITY & PRIVACY
RESEARCH GROUP

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

- Overview on web applications with databases
- SQL Injection attack on SELECT statement
- SQL Injection attack on UPDATE statement
- Countermeasures

Interacting with Database in Web Application



SQL Overview

- The ***SELECT statement*** is the most common operation on databases used to ***retrieve information*** from a database

```
mysql> select * from credential;
+----+-------+-------+--------+-------+----------+-------------+---------+-------+------------+------------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName   | Password                                 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+------------+------------------------------------------+
|  1 | Alice | 10000 |   8000 | 9/20  | 10211002 |             |         |       | alice      | fdbe918bdae83000aa54747fc95fe0470fff4976 |
|  2 | Boby  | 20000 |  30000 | 4/20  | 10213352 |             |         |       |            | b78ed97677c161c1c82c142906674ad15242b2d4 |
|  3 | Ryan  | 30000 |  50000 | 4/10  | 98993524 |             |         |       |            | a3c50276cb120637cca669eb38fb9928b017e9ef |
|  4 | Samy  | 40000 |  90000 | 1/11  | 32193525 |             |         |       |            | 995b8b8c183f349b3cab0ae7fccd39133508d2af |
|  5 | Ted   | 50000 |      1 | 11/3  | 32111111 |             |         |       | alicealice | 99343bff28a7bb51cb6f22cb20a618701a2c2f58 |
|  6 | Admin | 99999 | 400000 | 3/5   | 43254314 |             |         |       |            | a5bdf35a1df4ea895905f6f6618e83951a6effc0 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+------------+------------------------------------------+
```

SQL Overview

- The WHERE clause is used to ***set conditions*** for several types of SQL statements (e.g., SELECT, UPDATE, DELETE)

```
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***

SQL Overview

- The ***1=1*** predicate looks quite useless in real queries, but it will become useful in SQL Injection attacks
- We can use the UPDATE Statement to modify an existing record
- 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 credential; # comment till the end of the line
mysql> SELECT * FROM credential; -- comment till the end of the line
mysql> SELECT * FROM credential  /* in-line comment */ ;
```

Connecting to MySQL Database

SPRITZ
SECURITY & PRIVACY
RESEARCH GROUP

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

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

```
// Function to create a sql connection.
    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) {
        echo "</div>";
        echo "</nav>";
        echo "<div class='container text-center'>";
        die("Connection failed: " . $conn->connect_error . "\n");
        echo "</div>";
      }
      return $conn;
    }
```

- 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
    session_start();
    // if the session is new extract the username password from the GET request
    $input_uname = $_GET['username'];
    $input_pwd = $_GET['Password'];
    $hashed_pwd = sha1($input_pwd);
.....

// create a connection
    $conn = getDB();
    // Sql query to authenticate the user
    $sql = "SELECT id, name, eid, salary, birth, ssn, phoneNumber, address, email,nickname,Password
    FROM credential
    WHERE name= '$input_uname' and Password='$hashed_pwd'";
```

SQL Injection Attack

- Everything provided by user will become part 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 id, name, eid, salary, birth, ssn, phoneNumber, address, email, nickname, Password
    FROM credential
    WHERE name= '   ' and Password='   '
```

- The SQL statement will become the following

```
SELECT id, name, eid, salary, birth, ssn, phoneNumber, address, email, nickname, Password
    FROM credential
    WHERE name= 'alice' and Password='test'
```

SQL Injection Attack

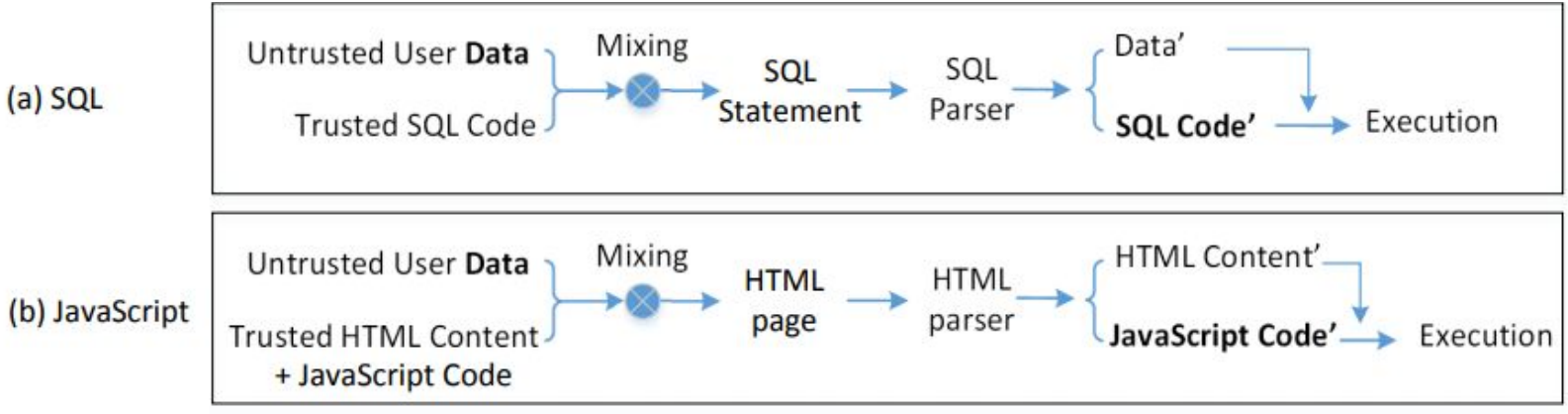
- If the statement is UPDATE or INSERT INTO, we can change the database

```
<?php
  session_start();
  $input_email = $_GET['Email'];
  $input_nickname = $_GET['NickName'];
  $input_address= $_GET['Address'];
  $input_pwd = $_GET['Password'];
  $input_phonenumber = $_GET['PhoneNumber'];
  $uname = $_SESSION['name'];
  $eid = $_SESSION['eid'];
  $id = $_SESSION['id'];


.....


 $sql = "UPDATE credential SET
nickname='$input_nickname',email='$input_email',address='$input_address',Password='$hashed_pwd',Ph
oneNumber='$input_phonenumber' where ID=$id;";
```

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

- 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***

```
Before encoding: aaa'
After encoding: aaa\'
```

- 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

```
<?php
$conn = new mysqli("localhost", "root", "seedubuntu", "dbtest");
$email = $mysqli->real_escape_string($_GET['Email']);
$pwd = $mysqli->real_escape_string($_GET['Password']);
....
 $sql = "SELECT  * from where ID="$email" and password="$pwd";
```

Countermeasures

- 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: 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


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

Using prepared statements, we separate code and data.

```
$conn = new mysqli ("localhost", "root", "seedubuntu", "dbtest");
$sql = "SELECT Name, Salary, SSN
        FROM employee
        WHERE eid= ? and password=?";
if ($stmt = $conn->prepare ($sql)) {
        $stmt->bind_param ("ss", $eid, $pwd);
        $stmt->execute();
        $stmt->bind_result ($name, $salary, $ssn);
```

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.