Exercise 4: Web Security in Practise

Introduction:

For this exercise, we have selected to develop a small web application that has SQL vulnerability. It is implemented using Express.js framework.

Vulnerability in code:

```
router.get('/', function(request,response){
    response.sendFile(__dirname + "/view/login.html");
});
router.post('/login/', function(request,response){
    let email = request.body.email;
    let password = request.body.password;
    /**
     * Use of Stored Procedure
     */
    let query = "CALL login(?,?,@success,@sessionId);";

    /**
     * Use of prepared statement
     */
    sql.query(query,[email,password] ,function(err, res){
        var result = {};
        if(!err){
        if(res['affectedRows'] == 1){
            let query = "SELECT @success, @sessionId;";
            sql.query(query, function(err,res){
                if(!err){
                    if(res[0]['@success'] == 1){
                        result['success'] = true;
                        result['sessionId'] = res[0]['@sessionId'];
                        result['message'] = "Authentication Successful!"
                        response.json(result);
```

Abstracts the logic of login into a Stored Procedure

Use of prepared statement makes the SQL call secure from SQL injection

Adding checks on the response from query before sending it to client prevents from data leak

```
                }else if(res[0]['@success'] == 0){
                    result['success'] = false;
                    result['message'] = "Authentication Failed! No user with"
                                + "given email and password exists!!"
                    response.json(result);
                }
            }

        });
    }
  }else{
    result['success'] = false;
    result['message'] = err;
    response.json(result);
  }
  });
});

module.exports = router;
```

Stored Procedure used in above code snippet for login:

```
create
    definer = root@localhost procedure login(IN emailid varchar(30),
                                             IN pwd varchar(100),
                                             OUT success tinyint,
                             OUT session_id varchar(100))
BEGIN
SELECT id
INTO @user_id
FROM user
WHERE email = emailid and password = md5(pwd);

SELECT row_count() into @affectedRows;
IF @affectedRows = 1 THEN
  INSERT INTO `session`(`id`, `user_id`)
   VALUES (UUID_TO_BIN(UUID()),@user_id);
   SELECT row_count() into @rowsAdded;

   IF @rowsAdded = 0 THEN
     SET success = 0;

   ELSE
     SELECT BIN_TO_UUID(id)
      INTO @session_id
```

Checking number of rows returned and equating the count with helps us to confirm that the query returned **only one** record, increasing security.

```
        FROM session

        WHERE user_id = @user_id
        ORDER BY  created_on DESC
        LIMIT 1;

       SET success = 1;
       SET session_id = @session_id;
    END IF;
ELSE
                SET success = 0;

END IF;

END;
```

Below screenshots demonstrate the security fix mentioned below: