

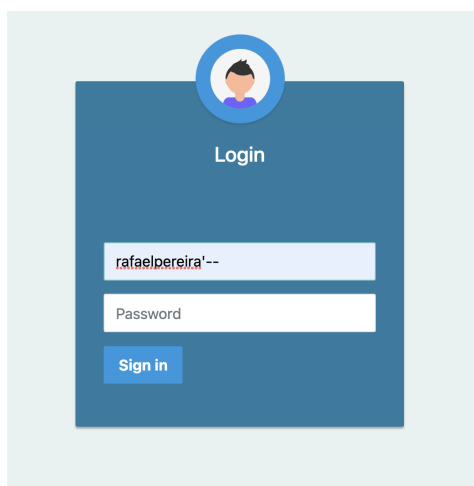
ANALYSIS

1. CWE-89: SQL Injection

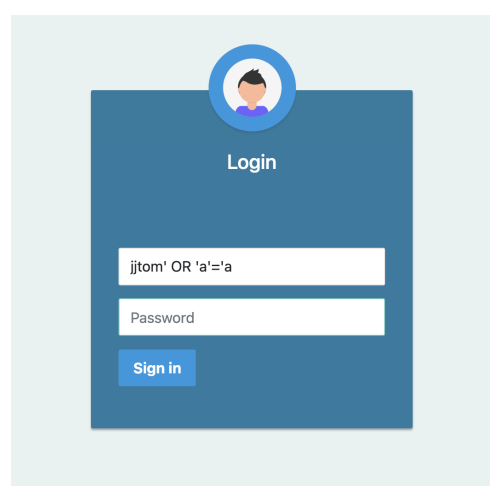
- *Vulnerable Implementation:*

SQL injection was exploited in user validation (login). The login input uses the information entered directly in the command that makes the request in the database, this makes that when we try to perform user authentication without a password, we can access the client's page. The following image shows the failed authentication implementation:

```
app.post('/login', (req, res) => {  
  sqlCommand1 = `SELECT * FROM student WHERE username='${req.body.username}' and password='${req.body.password}'`;   
  sqlCommand2 = `SELECT * FROM teacher WHERE username='${req.body.username}' and password='${req.body.password}'`;   
  
  let validate = false  
  
  db.all(sqlCommand1, function(err,rows){  
    rows.forEach(function (row) {  
      if(row.username && row.password){  
        validate = true;  
        studentLogin = true;  
        req.session.username = row.username;  
        req.session.name = row.name;  
        req.session.number = row.number;  
        req.session.saldo = row.saldo;  
        req.session.qteacher = qteacher;  
        req.session.fteacher = fteacher;  
        req.session.mteacher = mteacher;  
        res.render('student',{name:req.session.username,number:req.session.number,saldo:req.session.saldo,qteacher:req.session.qteacher,fteacher:req.session.fteacher,mteacher:req.session.mteacher});  
      }  
    });  
  });  
  if(!validate){  
    db.all(sqlCommand2, function(err,rows){  
      rows.forEach(function (row) {  
        if(row.username && row.password){  
          validate = true;  
          studentLogin = false;  
          req.session.username = row.username;  
          req.session.name = row.name;  
          req.session.subject = row.subject;  
          res.render('teacher',{name:req.session.name, subject:req.session.subject});  
        }  
      });  
      if(!validate) res.end('Invalid username or password');  
    });  
  }  
})  
});
```



A login form with a blue header and a circular profile icon. The title "Login" is centered. Below the title, there are two input fields: the first contains the text "rafaelpereira--" and the second is labeled "Password". At the bottom, there is a blue button labeled "Sign in".



A login form with a blue header and a circular profile icon. The title "Login" is centered. Below the title, there are two input fields: the first contains the text "jijtom' OR 'a'='a" and the second is labeled "Password". At the bottom, there is a blue button labeled "Sign in".

- *Secure Implementation:*

Na implementação segura, o que fizemos foi garantir que o username e a password estavam na base de dados, e não verificá-los individualmente, logo, ao tentar entrar da

mesma forma que na versão vulnerável, o invasor é direcionado a uma página de erro. Segue o código correto:

```
// login route
app.post('/login', (req, res) => {

  sqlCommand1 = `SELECT * FROM student WHERE username= ? and password=?`, [req.body.username, req.body.password];
  sqlCommand2 = `SELECT * FROM teacher WHERE username= ? and password=?`, [req.body.username, req.body.password];

  let validate = false

  db.all(sqlCommand1, [req.body.username, req.body.password], function(err, rows){
    rows.forEach(function (row) {
      if(row.username == row.password){
        validate = true;
        studentLogin = true;
        req.session.username = row.username;
        req.session.name = row.name;
        req.session.number = row.number;
        req.session.saldo = row.saldo;
        req.session.qteacher = teachers['qteacher'];
        req.session.fteacher = teachers['fteacher'];
        req.session.mteacher = teachers['mteacher'];
        res.render('student', {name: req.session.username, number: req.session.number, saldo: req.session.saldo, qteacher: req.session.qteacher, fteacher: req.session.fteacher, mteacher: req.session.mteacher});
      }
    });
  });

  if(!validate){
    db.all(sqlCommand2, [req.body.username, req.body.password], function(err, rows){
      rows.forEach(function (row) {
        if(row.username == row.password){
          validate = true;
          studentLogin = false;
          req.session.username = row.username;
          req.session.name = row.name;
          req.session.subject = row.subject;
          res.render('teacher', {name: req.session.name, subject: req.session.subject});
        }
      });
    });
    if(!validate) res.end('Invalid username or password');
  }
});
});
```

2. CWE-79: Cross Site Scripting

- *Vulnerable Implementation:*

This vulnerability can be noticed in the section for questions to the teacher, where it is possible to send, in addition to a text with the question itself, a script, link, images and items of any nature read by a web page. This happens because the text is inserted and read from the database without any treatment, therefore, if the user inserts an HTML code in the text box, it will be placed in the database and later allocated directly in the html of the teacher's page.

Doubts

<script>alert("Hello teacher")</script>

- ☐ Renan Descalço
- ☐ Alberto Astuto
- ☒ João José Tomate

Submit

```

app.post('/question', (req, res) => {
  const text = req.body.comment;
  let teacher = teachers[req.body.teacher];
  console.log(teacher);

  const sqlCommand = `INSERT INTO questions (username, content, teacher) VALUES ('${req.session.username}', '${text}', '${teacher}');`
  db.run(sqlCommand);
  console.log(sqlCommand);
  res.redirect('/');
});

app.get('/question', (req, res) => {
  if(studentLogin) {
    res.send("Access denied");
  } else {
    sqlCommand = `SELECT * FROM questions WHERE teacher='${req.session.name}';`
    console.log(sqlCommand);
    db.all(sqlCommand, (err, rows) => {
      if(err) {
        res.send("Not found");
      } else {
        let com = "<div><h1>Questions for : " + req.session.name + "</h1><br><br>\n"
        rows.forEach(function(row) {
          console.log(row);
          const username = row.username;
          const text = row.content;
          com += "<h3>Student: " + username + "</h3><br>" + text + "<br>";
        });
        com += "</div>"
        res.render('questions', {result:com});
      }
    });
  }
});
}
});

```

- *Secure Implementation:*

To prevent Cross site scripting we used two different methods such as: Trim() for trimming chars at the beginning and end of the string and escape() to replace non chars for '/' and respective HTML entries. All of the methods were used from express-validator package.

```

app.post('/question', [check('comment').trim().escape()], (req, res) => {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    console.log("Text input error!")
    res.redirect('/')
  }
  else{
    const text = req.body.comment;
    let teacher = teachers[req.body.teacher];
    console.log(teacher);

    const sqlCommand = `INSERT INTO questions (username, content, teacher) VALUES ('${req.session.username}', '${text}', '${teacher}');`
    db.run(sqlCommand);
    console.log(sqlCommand);
    res.redirect('/');
  }
});

```

3. CWE-434: Unrestricted Upload of File with Dangerous Type

- *Vulnerable Implementation:*

In this case, we are dealing with the section where the student can send a file with questions to the teacher. In the vulnerable version, the user can send any type of file, malicious or not, as there is no validation of what type of file he is sending.

```
<div class="send-image">
  <h2>Doubts file</h2>
  <form action="/question" method="POST">
    <div class="form-group">
      <!-- <input name="teacher" class="form-control" placeholder="Nome do professor" style="width: 30vw;margin-bottom: 15px;" -->
      <form action="/action_page.php">
        <input type="file" id="myFile" name="filename" >
        <input type="submit">
      </form>
    </div>
    <p> </p>
    <input type="radio" id="mteacher" name="teacher" value="mteacher">
    <label for="mteacher"><%-mteacher%></label><br>
    <input type="radio" id="fteacher" name="teacher" value="fteacher">
    <label for="fteacher"><%-fteacher%></label><br>
    <input type="radio" id="qteacher" name="teacher" value="qteacher">
    <label for="qteacher"><%-qteacher%></label>
  </div>
</form>
</div>
```

- *Secure Implementation:*

In the secure version, only image files(JPEG,JPG,PNG,GIF) and text files can be uploaded.

```

<body>
  <!-- File input field -->
  <p>Upload an Image</p>
  <input type="file" id="file"
    onchange="return fileValidation()" />

  <!-- File Validation-->
  <div id="imagePreview"></div>
  <script>
    function fileValidation() {
      var fileInput =
        document.getElementById('file');

      var filePath = fileInput.value;

      //Allowed file types
      var allowedExtensions =
        /\.(jpg|\.jpeg|\.png|\.gif)$/i;

      if (!allowedExtensions.exec(filePath)) {
        alert('Invalid file type');
        fileInput.value = '';
        return false;
      }
    }
  </script>
  <button type="submit" class="btn btn-primary" style="background:#1b98e0 !important;border: #1b98e0;">Submit</button>
</body>

```

4. CWE-20: Improper Input Validation

- *Vulnerable Implementation:*

This vulnerability is implemented on the student's tuition fee payment page. In a faulty version, the user can enter negative amounts in the payment section and instead of his balance decreasing, it is actually increased, a fact that happens due to the non-verification of the entry amount, that is, if it is negative or not.

```

app.post('/propinas', (req, res) => {
  sqlCommand1 = `Select saldo from student WHERE number='${req.session.number}'`;

  var saldo = 0;

  db.all(sqlCommand1, (err, rows) => {
    rows.forEach(function(row) {
      saldo = row['saldo'];
      const payment = req.body.pay;
      saldo -= payment;
      sqlCommand2 = `UPDATE student SET saldo=${saldo} WHERE number='${req.session.number}'`;
      db.run(sqlCommand2);
      req.session.saldo = saldo;
    });
  });

  res.redirect('/');
});

```

- *Secure Implementation:*

To ensure that this error would not happen, a validation of the input value was implemented, ensuring a positive value

```
app.post('/propinas', (req, res) => {
  sqlCommand1 = `Select saldo from student WHERE number='${req.session.number}'`;

  var saldo = 0;

  db.all(sqlCommand1, (err, rows) => {
    rows.forEach(function(row) {
      saldo = row['saldo'];
      const payment = req.body.pay;
      if(payment < 0){
        console.log("Erro: negative value");
      }else{
        saldo -= payment;
        sqlCommand2 = `UPDATE student SET saldo=${saldo} WHERE number='${req.session.number}'`;
        db.run(sqlCommand2);
        req.session.saldo = saldo;
      }
    });
  });

  res.redirect('/');
});
```

5. CWE-2087: Improper Authentication

- *Vulnerable Implementation:*

Each teacher, if logged in to their account, must be able to see the assignments of all students in a given school year - a fact that is not allowed if the user is a student. Vulnerability CWE-2087 can be observed in the wrong implementation of our program when it is enough that the user has the link of the assignments page of a certain year to be able to see the data displayed by it. This happens, because the validation of what type of user makes the request to access that route is not done.

```
app.get('/pautas', (req, res) => {
  let sqlCommand = `SELECT * FROM student WHERE classroom='${req.query.ano}'`;

  db.all(sqlCommand, (err, rows) => {
    rows.forEach((row) => {
      let com = "<div><ul>\n"
      let name = row.name;
      const number = row.number;
      const gradeM = row.gradeM;
      const gradeP = row.gradeP;
      const gradeC = row.gradeC;
      com += "<li>" + name + " | " + number + " | " + gradeM + " | " + gradeP + " | " + gradeC + "</li>";
      com += "</ul></div>";
      res.render('pautas', {result:com});
      numbers=[];
    })
  });
});
```

- *Secure Implementation:*

To solve the problem, we performed the validation of the user who requests the guidelines. If you are a student, you do not have access and an error is displayed. On the other hand, if you are a teacher, the class agenda is displayed.

```
app.get('/pautas', (req, res) => {
  let sqlCommand = `SELECT * FROM student WHERE classroom='${req.query.ano}'`;

  db.all(sqlCommand, (err, rows) => {
    if(studentLogin == false){
      rows.forEach((row) => {
        let com = "<div><ul>\n"
        let name = row.name
        const number = row.number;
        const gradeM = row.gradeM;
        const gradeP = row.gradeP;
        const gradeC = row.gradeC;
        com += "<li>" + name + " | " + number + " | " + gradeM + " | " + gradeP + " | " + gradeC + "</li>";
        com += "</ul></div>";
        res.render('pautas', {result:com});
        numbers=[];
      })
    }else{
      res.end('erro');
    }
  });
});
})
```