Sanitisation

Welcome to this lab activity

In this lab activity, you will implement sanitisation in your Node.js application. This essential technique will protect your application from malicious attacks.

Task 1: Make a copy of your previous lab

You will start with your previous lab code.

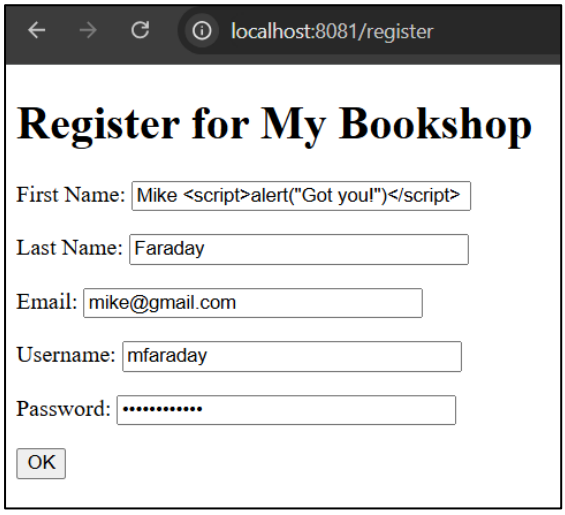
1. Make a copy of the folder `26_validation` from your previous lab and call it `27_sanitisation`.
2. Open the new folder in Visual Studio Code.
3. Run the code and confirm that it works as expected.

Task 2: Demonstrate cross-site scripting

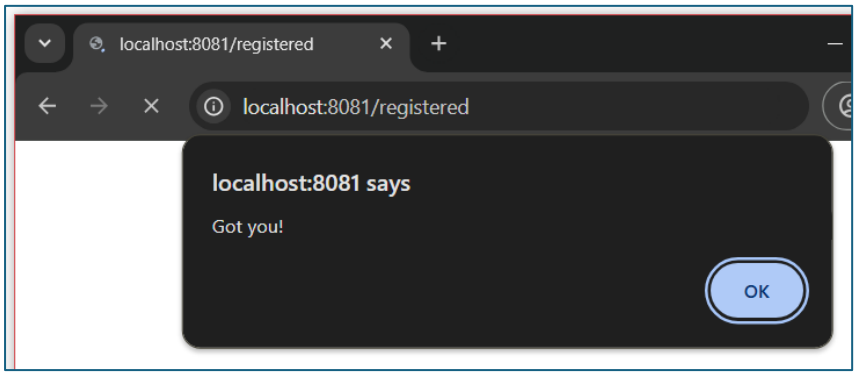
Your register page is currently vulnerable to a type of injection attack called cross-site scripting or XSS. This allows a malicious user to inject client-side JavaScript into your application via an unprotected form. Let's demonstrate this before we fix the vulnerability.

4. Go to the register form and fill in some details. In the First Name field, add a simple client-side JavaScript script to display an alert message:

```
Mike <script>alert("Got you!")</script>
```



5. Press OK on the form. You should see an alert in your browser:



6. Look through your code and consider what exactly happened here.

In this case, the injected script `<script>alert("Got you!")</script>` was quite innocent, just displaying a message on the browser. But potentially more serious attacks can be implemented, such as session hijacking, where an attacker steals a user's cookies.

You will now add sanitisation to your application to prevent such attacks.

Task 3: Add sanitisation

7. You will need a module called express-sanitizer. Install it using the following command from a terminal session:

```
npm install express-sanitizer
```

8. Then require it in your index.js file:

```
const expressSanitizer = require('express-sanitizer');
```

9. Add the following line of code to your index.js file:

```
// Create an input sanitizer

app.use(expressSanitizer());
```

10. Sanitise the firstname field using the req.sanitize() function. You need to sanitise the value retrieved from the form everywhere it is used. The easiest way to do this is to assign the sanitised value to a variable and use that variable in the rest of the code. The code in yellow below should be added or changed in your code:

```
router.post("/registered", [check('email').isEmail()], (req, res, next) => {
    // Validation
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
```

```
        res.send("Invalid email address");
        return;
    }

    // Sanitisation
    const firstname = req.sanitize(req.body.first);

    const saltRounds = 10
    const plainPassword = req.body.password

    // Hash the password
    bcrypt.hash(plainPassword, saltRounds, function (err, hashedPassword) {
        // Store hashed password in your database.
        let sqlquery = "INSERT INTO users (username, firstname, lastname,
hashed_password) VALUES (?,?,?,?)"

        // Execute sql query
        let newrecord = [req.body.username, firstname, req.body.last,
hashedPassword]
        db.query(sqlquery, newrecord, (err, result) => {
            if (err) {
                next(err)
            } else {
                result = 'Hello ' + firstname + ' ' + req.body.last + ' you are now
registered!  We will send an email to you at ' + req.body.email
                res.send(result)
            }
        })
    })
});
```

11. Repeat the XSS attack that you performed in task 2. Did the attack work this time?

Task 4: Explore further

When tackling these lab activities, it's always good to stretch yourself by doing some research and attempting some changes on your own.

What other fields in your application need sanitisation? Apply sanitisation to these too.

End of lab

Congratulations on completing this lab.

You have successfully implemented sanitisation in your web application!