Security report

Notebridge group 2

In this report we will quickly go over all the safety precautions we have taken with our web application to make sure that it is as safe as possible. We have implemented hashed passwords for saving passwords in a more secure way, prepared statements to try and prevent sql injections by not giving users the ability to access the database outside some limited input and input sanitization to further help to make sure that user input does not get mixed with the code. We will explain all these methods separately in the sections below.

Hashed passwords

For saving our passwords in the database we are using salted hashed passwords. After the user inputs their password while registering a new account it will be sent to the server side, after this the password will be hashed. The way this happens is that a random 16 byte array is created as a salt to append to the passwords. These strings together are then hashed to make a secure way to store passwords where even users with the same passwords do not have the same hash. We store the hashed password and the salt in the database so that we can compare the password that is being inputted together with the salt from the database to see if it matches the correct password when logging in. While this way of saving passwords is secure, we still advise users to have a strong password, we also have checks for this, which for now only include a minimum password length for testing purposes, but we will also add other requirements to this, like special characters and things of that sort.

Prepared statements

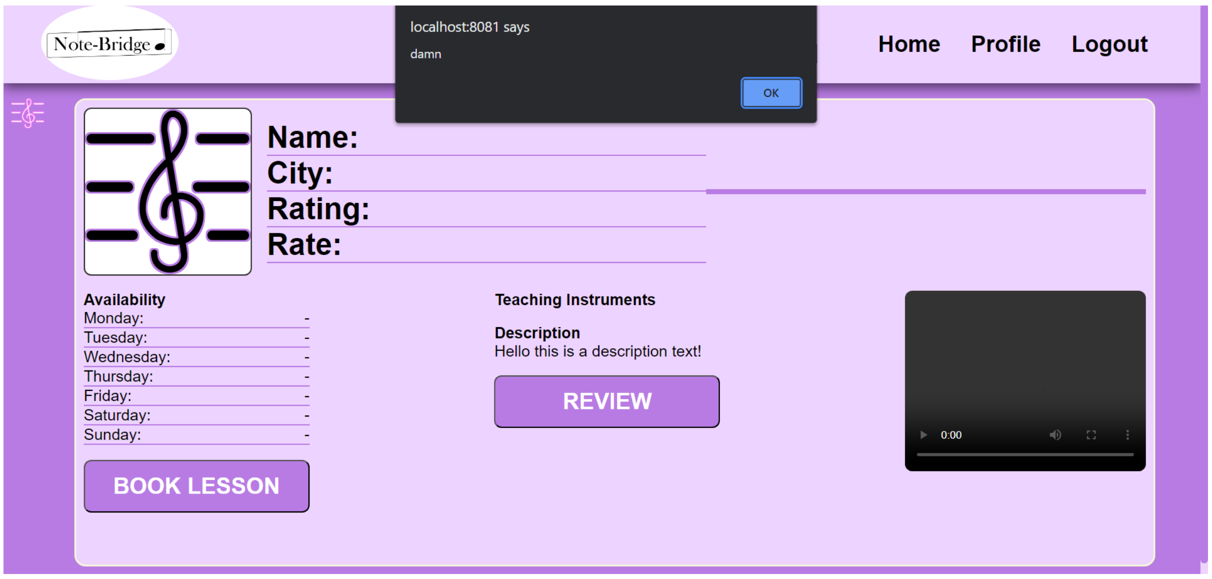
Our web application also makes use of prepared statements in the methods where we are inputting or querying data in and from the database. Prepared statements are pre compiled SQL queries that are parameterized and can be used multiple times. This not only protects our application against sql injections, it also has a bunch of other benefits that we will explain now. With the use of prepared statements the only input we need from the user is some parameters to fill in, this not only makes it safer and easier to work with, but the methods also become way easier to reuse for different queries, easier to maintain and troubleshoot if needed and maybe most importantly, it can also speed up our service. This is because we would only have to write and optimize the query once instead of constantly. So our prepared statements help us by making the back-end easier to work with, making the methods easily reusable, easily maintainable, easily optimizable, and most importantly, making our web application as safe as possible.

Input sanitization

Our web application, uses input sanitization to prevent vulnability exploits such as XSS (cross site scripting). Input sanitization is a process of validating and cleaning up user inputs to prevent potentially harmful or maliciouts content before it is used in a software application.

Cross site scripting or XSS exploits vulnerabilities by injecting malicious code in a form of a script into a web application, the code is then executed to unsuspecting users visiting the affected page. XSS could lead to different exploits such as unauthorized access, data theft, cookie hijacking, etc.

Parts of our web application that are vulnerable to XSS are our input fields, mainly those which inputs are stored in the database, or which the output can be accessed through URL parameters. These includes search bars, review section and account registration and login. Parts of the website have been tested and some are indeed vulnerable to XSS. A part that was tested and indeed vulnerable was the review section.


localhost:8081 says

damn

Name:
City:
Rating:
Rate:

Availability	
Monday:	-
Tuesday:	-
Wednesday:	-
Thursday:	-
Friday:	-
Saturday:	-
Sunday:	-

Teaching Instruments

Description
Hello this is a description text!

From Figure 1, it can be seen that a script was able to run when accessing a profile page that was injected with a script in the review section. The following script was written as a review for the corresponding person, the script will run everytime the user page is loaded. The script that's used to exploit the vulnerability was [<img src onerror="alert(*'message'*)"]. To prevent this some methods and testing was done.

The first solution that was tested was changing the innerHTML with innertext. InnerHTML renders complete markup and not just text. By changing it, running scripts would not be possible, but since it's only a text content, some functionalities of the website would not work properly. Instead sanitizing the input by removing or changing disallowed markups and validation. removing characters such as (`<`, `>`, `'`, `"`) or changing them with their corresponding HTML entity equivalent (`&lt`, `&gt`, `&$39`, `&quot;`).