Third year BTech Students’ Cybersecurity Projects-2022, Vishwakarma Institute of Technology, Pune, INDIA.

Design & Development of an intentionally vulnerable web application for enactment of SQL injection & Cross Site Scripting vulnerabilities: HackerLearn

By

Sakshi Patil (11910968), Shaunak Deshpande (11911180), Shrawani Shinde (11911031), Shruti Yadnik (11911408).

---------------------------------------------------------------------\*\*\*-------------------------------------------------------------------

***Abstract — In today's world, every human being relies on the internet for their basic to complex needs. Because the internet provides users with a great amount of information, its accessibility is essential. Major objectives of security are availability, integrity, and confidentiality. Cross-site scripting (XSS) and SQL Injection Attack (SQLIA) are two common and serious security flaws in web applications and databases. According to our research, there is not a specific engaging platform for cybersecurity students to practice these critical attacks. The proposed technology allows students to rehearse these attacks and gain a better understanding of the situation.***

***Keywords*** *— Security, Cyber Attacks, SQL Injection, XSS Attack*

1. INTRODUCTION

Cybersecurity is important because it protects all categories of data from theft and damage. Information systems used by the government and business sectors as well as sensitive data, personally identifiable information (PII), protected health information (PHI), personal information, and intellectual property data are all included. Your company can't defend itself against data breach campaigns without a cybersecurity program, making it an easy target for hackers.

Due to increased global connection and the use of cloud services like Amazon Web Services to hold private and sensitive data, both inherent risk and residual risk are rising. The likelihood that your firm may experience a successful cyber attack or data breach is increasing as a result of widespread inadequate cloud service configuration and highly skilled cybercriminals. Business executives cannot exclusively rely on standard cybersecurity tools like firewalls and antivirus software because hackers are growing more cunning and their strategies are becoming more resistant to traditional cyber defences. To stay well-protected, it's crucial to cover all aspects of cybersecurity.

Many cyber attacks are opportunistic, with hackers spotting vulnerabilities in a computer system’s defenses and exploiting them. This may involve finding flaws in the code of a website that allows them to insert their own code and then bypass security or authentication processes. It could also mean they install ‘malware’ – software which is specifically designed to damage a system – via a vulnerable third party site.Although terminology such as ‘cyber attacks’ and ‘hackers’ may conjure up images of sophisticated teams of computer experts with high-tech equipment, poring over lines of code, the reality is often quite different. Cyber attacks are much more likely to occur through mundane errors like a user choosing an easy-to-guess password or not changing the default password on something like a router.Cyber attacks are usually either criminally or politically motivated, although some hackers enjoy bringing down computer systems a thrill or sense of achievement.

Politically motivated cyber-attacks may occur for propaganda reasons, to harm the image of a particular state or government in the minds of the public. It might also have more pernicious intent, such as to leak sensitive intelligence, private communications or embarrassing data.

Cyber-attacks could potentially go even further, for example, government-backed hackers could theoretically create software to corrupt and destroy a weapons program, or other crucial infrastructure.

In the present world of technology and innovation, the best way to build cyber security experience is to practice. The main concern of our project is to develop a website where users can practice different cyber-attacks like SQL Injection, XSS.

1. **LITERATURE REVIEW**

SQL injection is a technique that exploits a security vulnerability occurring in the database layer of an application. The attack takes advantage of poor input validation in code and website administration. It allows attackers to obtain unauthorized access to the back-and database to change the intended application generated SQL queries. Researchers have proposed various solutions to address SQL injection problems. However, many of them have limitations and often cannot address all kind of injection problems. What’s more, new types of SQL injection attacks have arisen over the years. To better counter these attacks, identifying and understanding existing techniques are very important. In this research, we cover every kind of SQL injection attack as well as several methods and tools for detecting or avoiding them.

In [1] The PHP scripting language and Apache XAMPP Server were used to implement the suggested technique. The Apache XAMPP Server was chosen due to its cross-platform compatibility, it supports any operating system, and it also supports both PHP scripting language and SQL. PHP was chosen as a scripting language because it is the most popular server-side scripting language in building database-driven web-based applications.

One of the first vulnerabilities to be discovered is cross-site scripting. The user's browser receives an injection of malicious code from the source. This may lead to the theft of cookies containing Personal Information, including login information and credit card information.[2]

SQL injection is one of the most lethal attacks that can be launched against any web application involving databases. 64% web applications worldwide are vulnerable to SQL injection attack improper input mechanisms.[3]

SQL injection and XSS can be prevented using input

sanitization. Two sanitization methods include blacklisting

and whitelisting. Blacklisting, as name suggests, listing out

the inputs which are flagged as malicious in nature. Usage

of ‘<script>’ tags or use of operators such as “=” etc is not

allowed as they could play with the logic of codes and

queries running in the backend. Whitelisting is the polar

opposite of Blacklisting. It refers to the list of

symbols/characters that can be allowed.[5]

**CYBER ATTACKS :**

Two attacks will be discussed in this paper .

## XSS(Cross Site Scripting)

Cross Site Scripting, or XSS, is a type of injection attack. The browser side script known as JavaScript is typically used to write the injected code. When input is not adequately vetted or encoded, this attack takes place. It enables the hackers to bypass the SOP (Same Origin Policy).

Cookies are frequently used on websites to temporarily store your login information so you don't have to keep logging in. But this is quite dangerous. As these cookies are easily accessible from the user side with the browser using JavaScript or any other browser in-built tools, utilities, etc., it would be advisable to keep a user logged for a brief length of time when inactive rather than saving login credentials in the form of cookies[2]. As a result, the confidentiality of your sensitive data may be at risk. It can be used to steal login information, get access to unauthorized data, pose as someone else, carry out unauthorized actions, etc.

## SQL injection

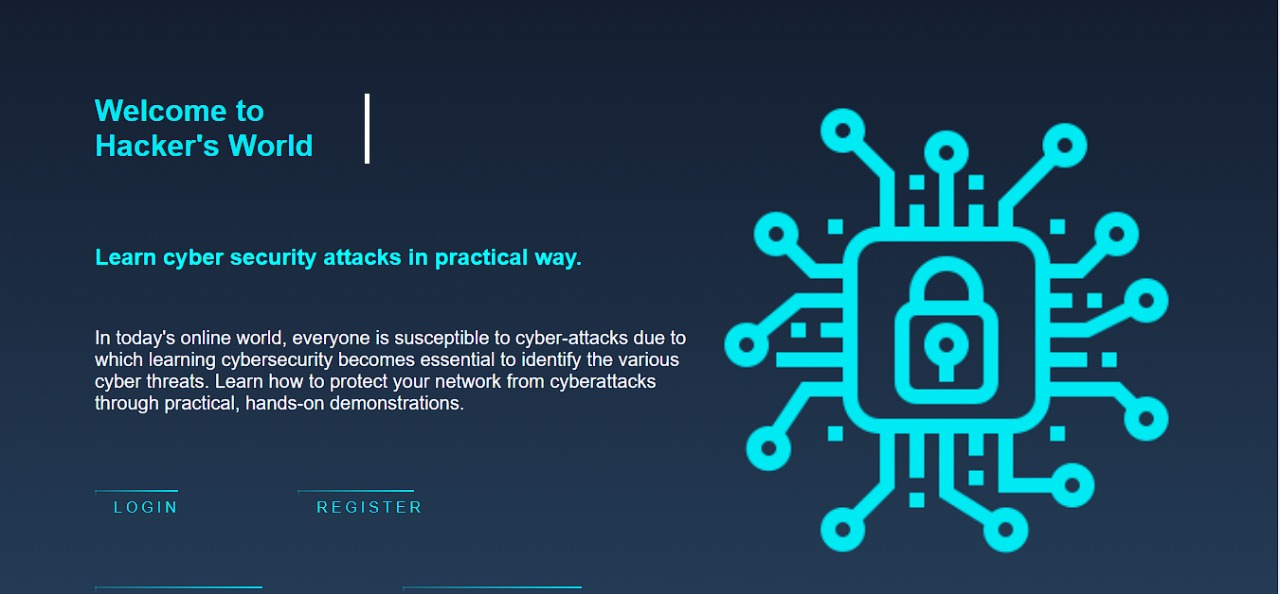
SQL injection is also an injection attack. As the name suggests, SQL code is injected into websites. Database vulnerabilities are exploited in this attack.

For most websites, data displayed on the web page arrives from SQL database servers. Login/ Registration forms, comments etc, everything is loaded into the backend from the databases. When we try to login to a website, an SQL query is run in the backend to verify your credentials. In case of registering an account, additional rows are added into the database. If sanitization is not practiced, input fields of the page could be filled with SQL code which could logically end up commenting out a certain section of the SQL query that is run in the backend which could result in granting login permission without knowing the actual credentials. This one of several examples of how SQL injection is implemented. It can be used to retrieve hidden data, subverting application logic, examining databases, UNION attacks(retrieve data from various databases) etc

1. **METHODOLOGY**

The project starts with a landing page which gives the user an option to register onto the website (for new users) or login to their account (for repeat users). The authentication system has been implemented using flask, flask-login and bcrypt. The hash of the password entered by the user gets stored during registration, and during login, the hash of the entered password is compared with the stored password. After login, the user is greeted by an eCommerce page, where the user can search for products. Along with that, there is also an option to open up the blog page of the eCommerce website, and leave a review for the products bought from the website. All this is setup so that this looks like an actual eCommerce business to the user. But the primary purpose of this web app is for practicing cyber security attacks in a secured environment. On the product search page, the user sees an input box, which can be used for practicing SQL injection attacks. The user can give malicious SQL queries into the search engine and try to steal user data. The user can also practice blind SQLI attacks here. Along with this, the blog and the product review section can be hit by Cross-Site Scripting (XSS) attacks. The backend is written in Flask, a Python-based microframework, which makes it lightweight, but also more susceptible to cyber threats, making it the perfect choice for our backend. The user can also perform Cookie stealing using the Cross-site scripting vulnerability in the web app. The SQL injection attack allows the user to get the passwords of the users, making it a good exercise in cracking Bcrypt hashes. Lastly, there is a logout button that the user can click on to log out of the session. The session management is done by the flask\_login module in python.

1. **RESULT AND DISCUSSION**



**Fig.1**. Home page

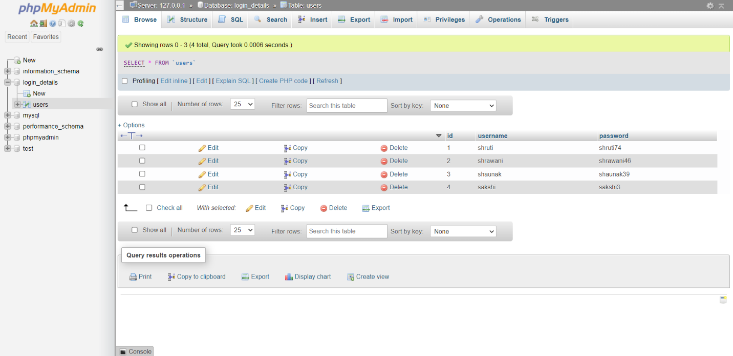
## SQL injection :

Let’s go to the login page now,

SQL injection is an attack used to take over database servers by dynamically poisoning SQL queries. In this attack, a special SQL query is passed as user input to the login page.



**Fig2**. Login Page for simulating SQL injection attack on database



**Fig3.** Login database



**Fig4.** After entering the correct username and password

 **Fig5**. After entering the Incorrect username and password

Later, when the backend code compiles the input along with the SQL query.  
The code then gets hijacked because these queries will change the meaning of the backend code.  
The SQL injection queries used for this demonstration project are:  
**' or 1=1--**  
This query will change all the SQL statement always true because of the or keyword.  
And the rest of the SQL statement after '--' will be commented.  
**admin' or '1'='1/ username' or '1'='1**

**** **Fig6.** The input along with the SQL query.

This query will change all the SQL statement will also always be true.  
Because of the or keyword, and the two-expression producing true values.



**Fig7**. The output along with the SQL query

We've already seen the injection part. Now we'll look at how to avoid this attack.

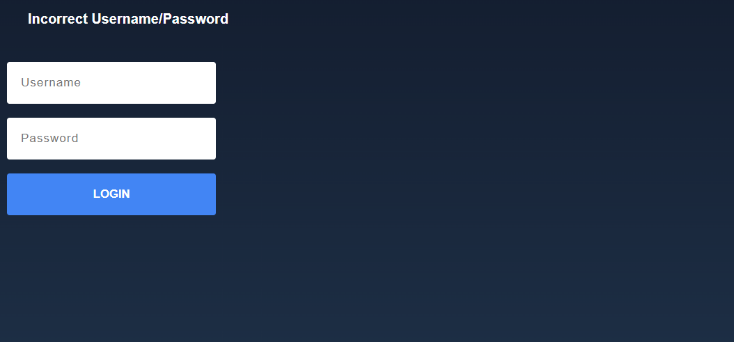
**Prevention Method:**

Now we need to add two lines of code to the backend PHP script.

**$uname = mysqli\_real\_escape\_string($conn, $uname);// admin or 1=1**

**$pass = mysqli\_real\_escape\_string($conn, $pass);**

Using these functions makes your site less vulnerable to SQL injection attacks, where an attacker puts SQL syntax into a form field to compromise your site. mysqli\_real\_escape\_string() "escapes" special characters so that MySQL interprets them as literal string characters rather than operators in the query.



**Fig8**. The output along with the SQL prevention query

Thus, SQL Injection and Prevention were the main concerns.

**SQL Attack on product page:**

Since the main use of the website is for SQL injection and Cross-Site Scripting, both these attacks were tried out on the website. The SQL Injection attack was simulated on a product page of the website. It includes an input button that takes a search parameter and returns the result based on the input which is entered.

Graphical user interface, website

Description automatically generated

**Fig9**. Product Page

Since this webpage takes user input, and displays a result based on that input, a malicious user could try and use an SQLI attack to try to gain useful information from here. In order to check if there is a SQL based database in the backend, and the database is susceptible to SQLI attacks, a part of an SQL query was entered into the search box. It was suspected that the backend will have a pattern matching based SQL query in the background, loosely matching the format: SELECT [some discerning information of the product] FROM [the name of the table] WHERE [name of the product] LIKE ‘[input\_string]’; where input\_string is the input given by the user. So a test query, “ ’;-- ” was input into the input box. The apostrophe closes the input string, making it an empty parameter; the semicolon acts as an ending character for the query, and the two hyphens act as the beginning of a comment, making sure that everything that follows it is not executed as code, essentially bypassing the query written by the backend developer. This is equivalent to querying the database for all of the products present in the product information table.

Graphical user interface

Description automatically generated

**Fig10**. Entire products table being shown on running a malicious query

As predicted, the website returns all the data in its product information table, confirming that it is susceptible to SQL injection. Next, another query was run on the server, to find out which vendor of SQL is running on the server. On execution of the “Apples' AND 0 = SLEEP(2);--” query, the following output occurs after 2 seconds.

*Graphical user interface, application

Description automatically generated*

**Fig11**. Data is returned after 2 seconds

This is because, the backend was tricked into returning the result of a simple search for Apples, and asked it to wait for two seconds before returning the output. The syntax for doing such a thing varies from vendor to vendor, and the executed query is the syntax for MySQL, a popular SQL-based database management system. Had it been some other DBMS, different syntax would have to be used, and the DBMS would be found out.

Another useful thing to check would be if the backend is capable of sending data other than that in the products table of the database. To check this, we perform a union operation using the query: “Apples' UNION (SELECT 1,2,3 from dual) ;--”.

Graphical user interface

Description automatically generated

**Fig12**. 1,2,3 get unionized with the output of the apples search query

Next, we can use the information\_schema database in MySQL, which has information about database metadata, to look further into the information in the entire MySQL server. To accomplish this, instead of performing a union with the dual table, we will perform a union with the TABLE\_NAME and TABLE\_SCHEMA columns in the tables table of information\_schema. The query to do this is: “Apples' UNION (SELECT TABLE\_NAME, TABLE\_SCHEMA, 3 from information\_schema.tables) ;--”

A screenshot of a computer

Description automatically generated with medium confidence

**Fig13**. TABLE\_NAME and TABLE\_SCHEMA data from information\_schema.tables

A picture containing graphical user interface

Description automatically generated

**Fig14**. (continued) TABLE\_NAME and TABLE\_SCHEMA data from information\_schema.tables

A picture containing graphical user interface

Description automatically generated

**Fig15**. (continued) TABLE\_NAME and TABLE\_SCHEMA data from information\_schema.tables

A screenshot of a computer screen

Description automatically generated with medium confidence

**Fig16**. . (continued) TABLE\_NAME and TABLE\_SCHEMA data from information\_schema.tables

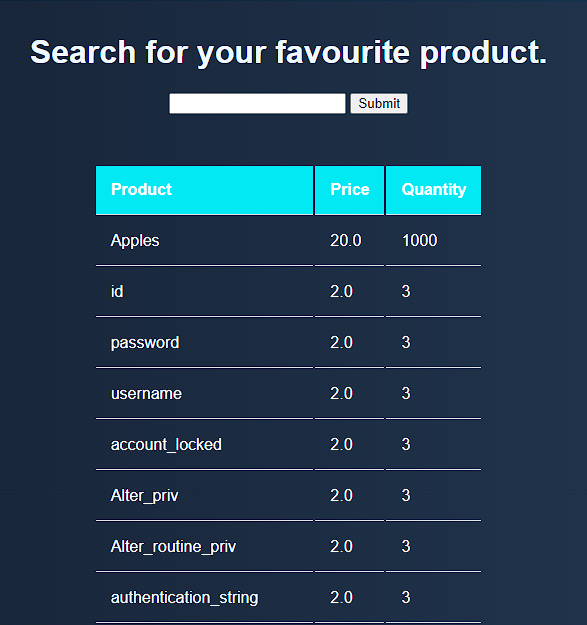
***A screenshot of a computer

Description automatically generated with medium confidence***

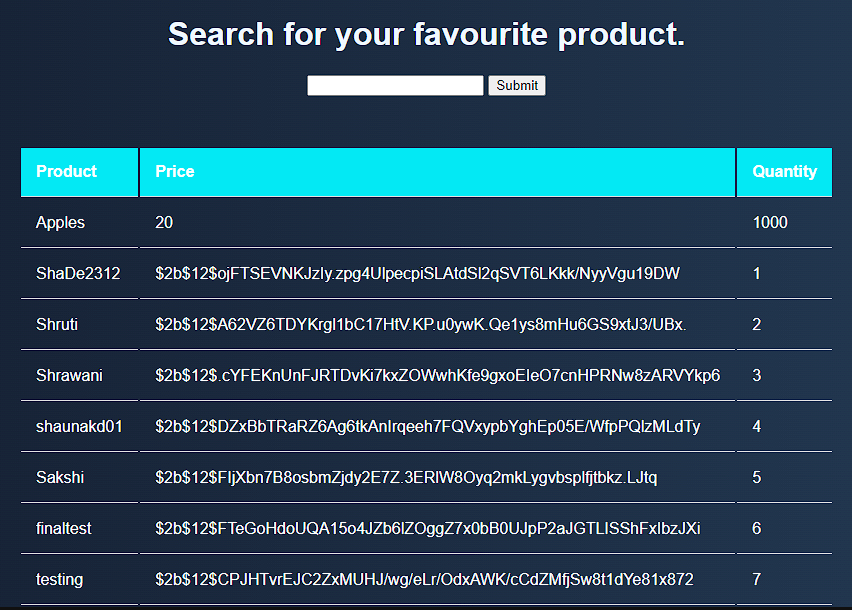
**Fig17**. TABLE\_NAME and TABLE\_SCHEMA data from information\_schema.tables. Notice the user,product,comments and product\_review tables from honours database

As you can see in the last image, we have a database called honours which has the tables user, product,comments, and product\_review. Here, the product table might not have any useful information, but the user table might have some important user identification information, such as usernames and passwords.

Since we don’t yet know the names of the columns of the user table, we will take the information from information\_schema.columns, using the query: “Apples' UNION (SELECT COLUMN\_NAME,2,3 FROM information\_schema.columns WHERE TABLE\_NAME='user') ;--”



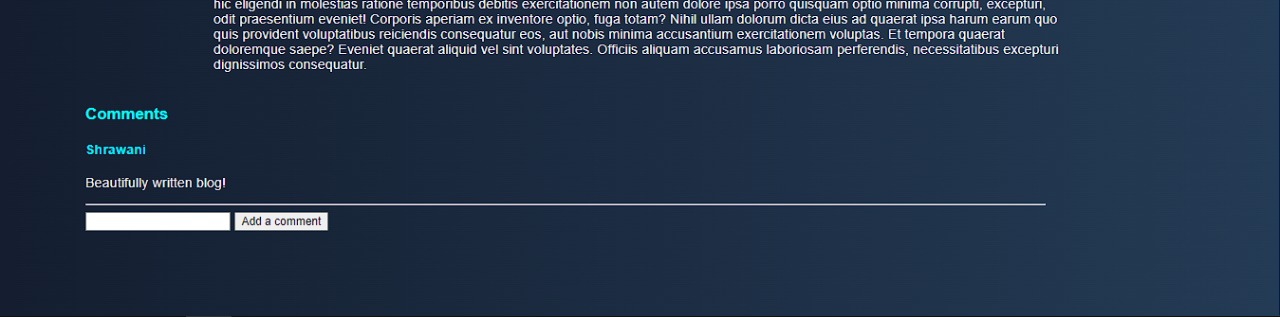
Here, we get the names of the columns: id, username and password. Now we can union this data with the Apples keysearch using the query: “Apples' UNION (SELECT username,password,id FROM honours.user );--”



**Fig20. The usernames, hashed passwords and ids of the users**

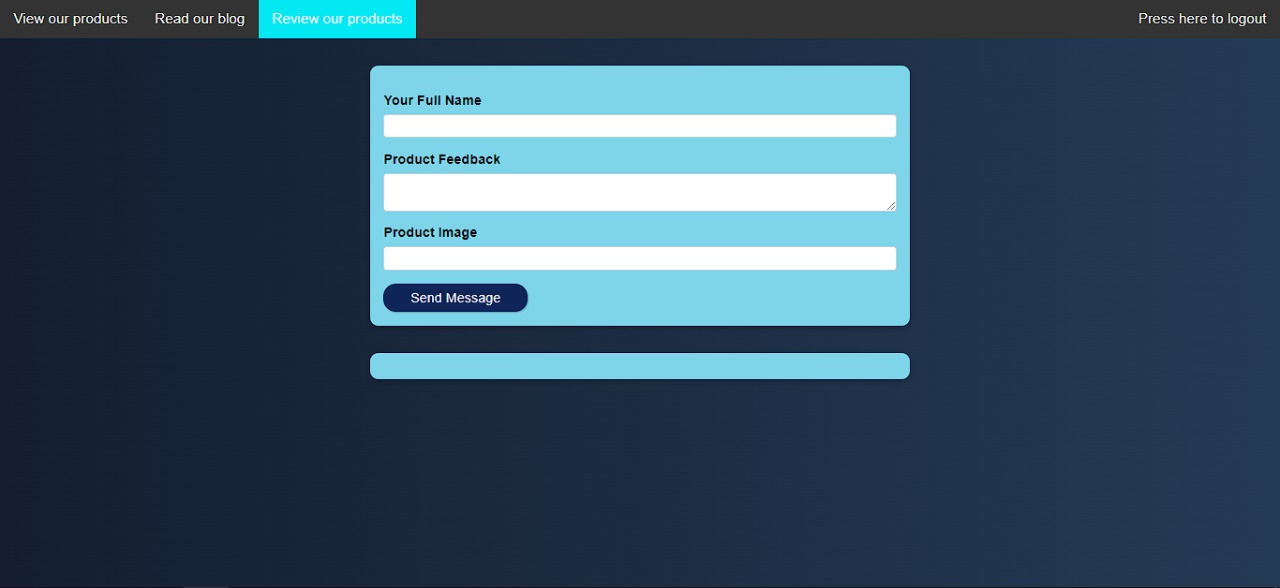
As we can see, we have the usernames, hashed passwords, and IDs of the users, which can be maliciously used for identity theft.

**Fig18**. The different column names of our user table, which were extracted from information\_schema.columns



**Fig19.**Webpage for simulating stored XSS attack on Blog comments

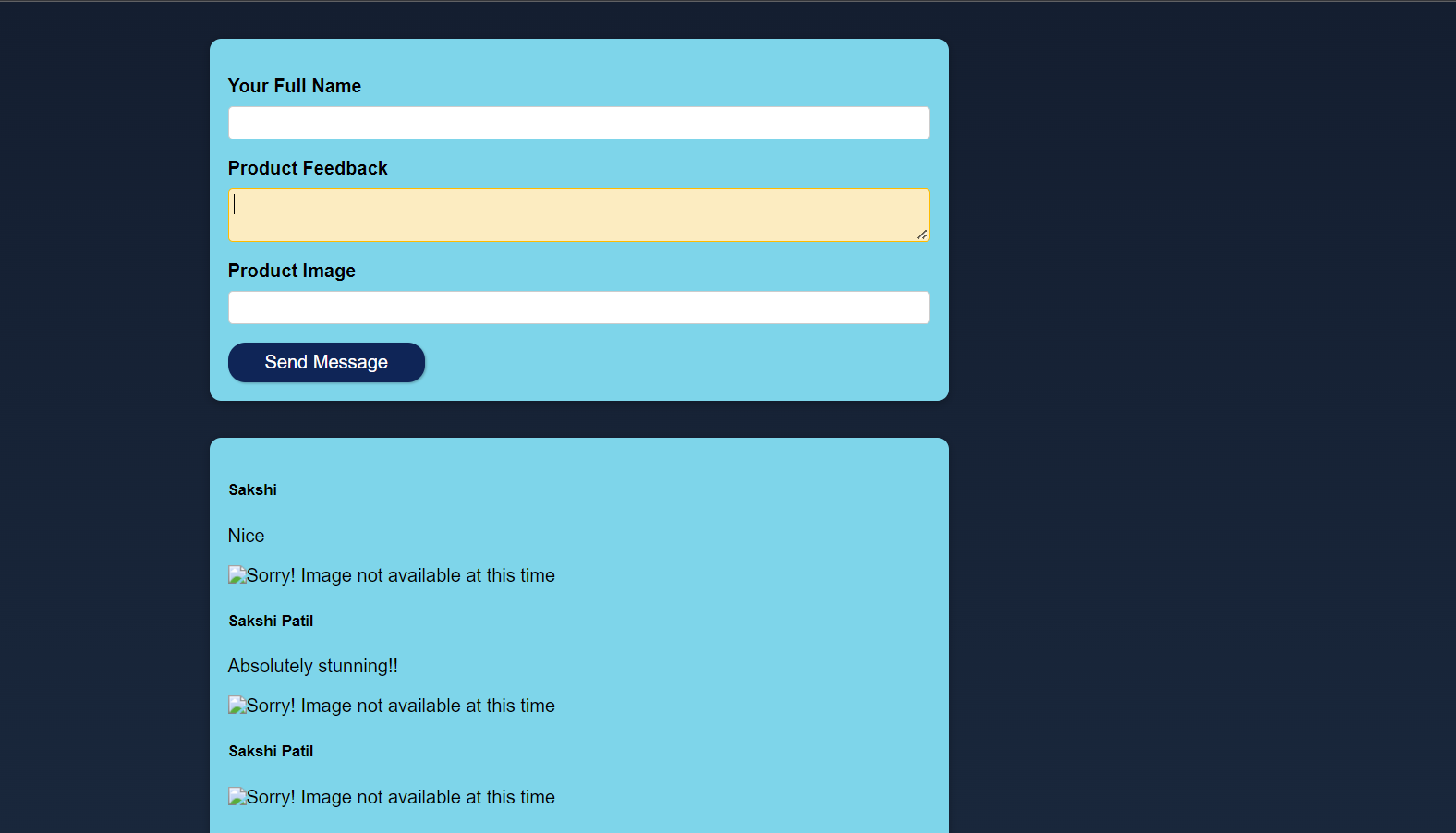
Cross site scripting attack on Product reviews page:



**Fig21.**Webpage for simulating stored XSS attack on Product

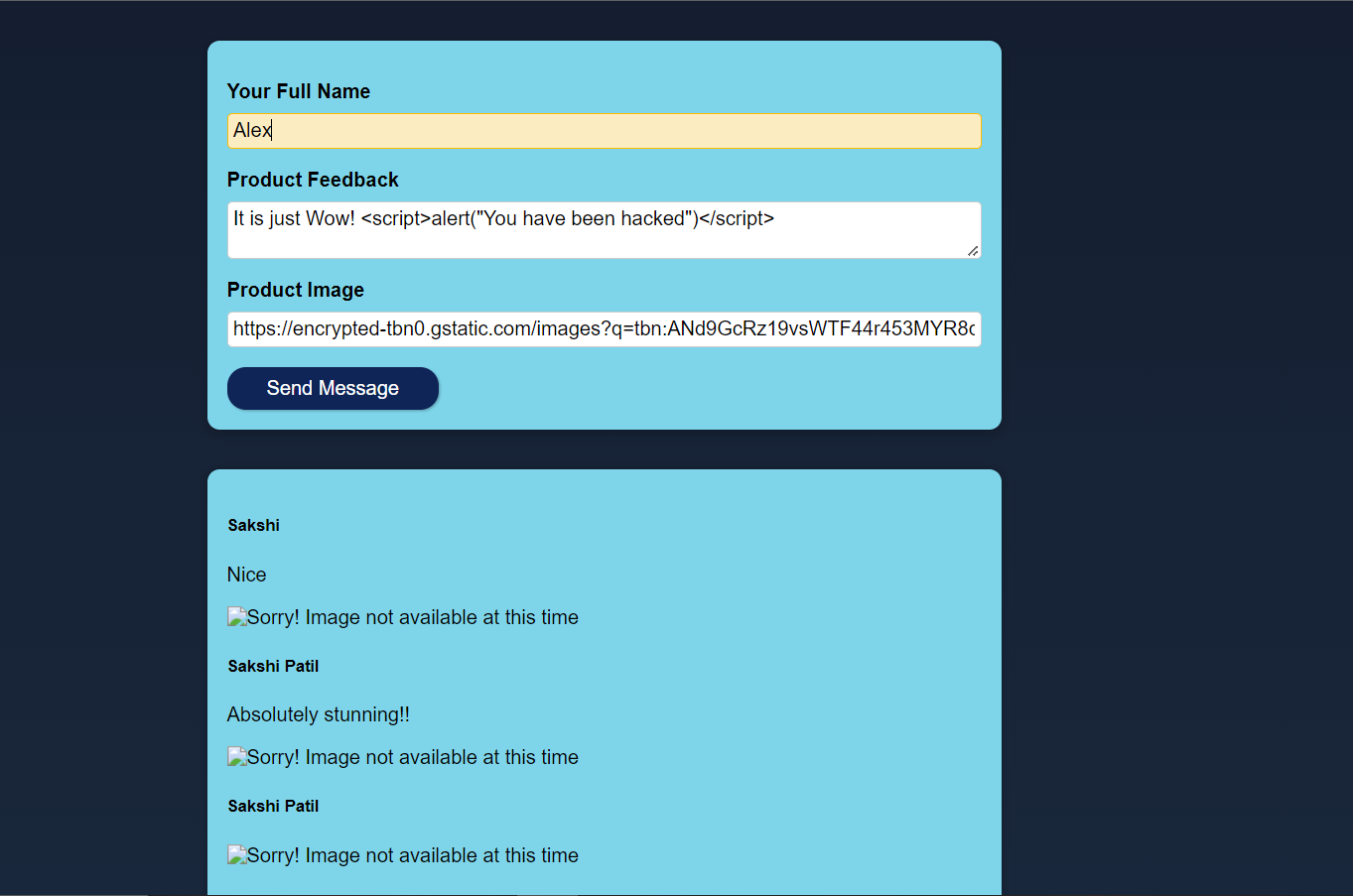
Reviews

A product review page has been made to implement stored XSS attacks. Since this project is about e-commerce website, a section for product reviews is a necessary component of every e-commerce business. User name, product feedback, and product image are the three input fields in this section of the product review. The information gathered from these input fields is saved in a MySQL database for that ‘Product\_Review’ table was created.



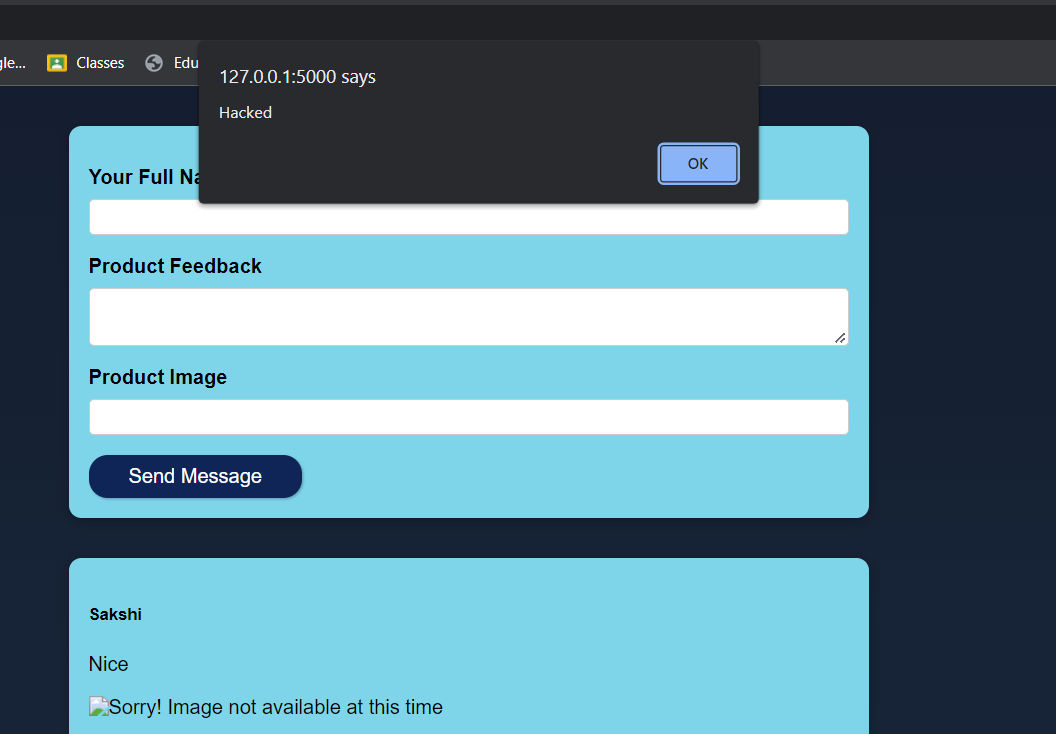
**Fig22.**Displaying previous product reviews given by the users

Our product review, which includes the user name, feedback, and image of the product, has been displayed after we entered some random relevant information in these input fields.



**Fig23.**Injecting malicious javascript in the feedback section.

Let's insert some malicious javascript code in the product feedback field to carry out a stored XSS attack.**“<script>alert(“Hacked!”);</script”**, this javascript has been injected along with some other text. After clicking on send message button, an alert window saying “Hacked” is popped up. This data is sent to the server and submitted to a database as our website is using database to store product reviews via HTTP requests.



**Fig24.**Alert window coming/ user has been infected

Here, an alert window that reads **"Hacked!"** has appeared. Each time a user accesses the page, their browsers download and execute the script in order to render the page and the alert window pops up.

Here, we have used very basic Java script code that merely causes an alert window to pop up as an example. But hacker can do many bad things to harm users. Hacker can steal the session cookies and perform session hijacking. Cross-site scripting vulnerabilities can typically be used by an attacker to pretend to be or pose as the victim user, execute any operation that the user is capable of, Read any information that the user has access to, obtain the user's login information, Inject trojan functionality into the web site and perform website vandalism virtually.

1. **FUTURE SCOPE**

The proposed application is capable of simulating SQL injection and XSS attack successfully. In future, we intend to include simulation functionalities of more cyber attacks such as CSRF, Man-in-the-middle, Zero-day attack, and so on, so that the web application can serve as a full-fledged cyber attack simulation platform.

1. **CONCLUSION**

SQL injection and cross-site scripting attacks are two of the most dangerous threats to web applications. The proposed web application serves as a simulation platform to practise attacks in a secured environment. It will assist students and enthusiasts interested in cyber security to obtain a better knowledge of how such cyber attacks occur.

1. **ACKNOWLEDGMENT**

We would like to express our gratitude to Prof. Anil Kadu, our project guide, for thoroughly educating us on all aspects of the project. We would also like to express our gratitude to our college, Vishwakarma Institute of Technology, for giving us the opportunity to work on and present a project at this level.

1. **REFERENCES**
2. Prakhar Tripathi , Rahul Thingla; ‘Cross site scripting (XSS) and SQL-injection attack detection in
3. web application’, International Conference on Sustainable Computing in Science, Technology & Management (SUSCOM-2019).
4. ‘XSS- What are Cross Site Scripting Attacks’- An article by Maximilian Schwarzmüller.

[3] Adit Bhosle, “Combination Attack: XSS+SQL Injection Attack Demonstration”, International Research Journal of Engineering and Technology (IRJET).

[4] Robinson, Memen Akbar, SQL Injection and Cross Site Scripting Prevention Using OWASP Web Application Firewall, International Journal of Informatics Visualization. VOL 2 (2018) NO 4.

[5] Muhammad Saidu Aliero, Imran GhaniAn algorithm for detecting SQL injection vulnerability using black-box testing,, Journal of Ambient Intelligence and Humanized Computing , 7th Febuary 2019.

[6] Robinson, Memen Akbar, SQL Injection and Cross Site Scripting Prevention Using OWASP Web Application

Firewall, International Journal of Informatics

Visualization. VOL 2 (2018) NO 4.

[7] Ouissem Ben Fredj, Omar Cheikhrouhou, An OWASP Top Ten Driven Survey on Web Application Protection Methods, Risks and Security of Internet and Systems, 15th international Conference, CRiSIS 2020, Paris, France, November 4-6, 202