Proceedings of the Fifth International Conference on Intelligent Computing and Control Systems (ICICCS 2021)
IEEE Xplore Part Number: CFP21K74-ART; ISBN: 978-0-7381-1327-2

Secure Web development using OWASP Guidelines

Shubham Kumar Lala
III Year CSE, School of Computer Science and Engineering, VIT University Chennai, Tamil Nadu, India
shubhamkumarlala.skl@gmail.com

Akshat Kumar
III Year CSE, School of Computer Science and Engineering, VIT University Chennai, Tamil Nadu, India
akshatkumar.dev@gmail.com

Dr. Subbulakshmi T.
Professor, School of Computer Science and Engineering, VIT University Chennai, Tamil Nadu, India
research.subbulakshmi@gmail.com

***Abstract*— Website security is a major concern for large organizations as well as individual developers, the rarer the technology used the harder it becomes to come up with secure practices for developing a website. Vulnerabilities that are not fixed during development, and are deployed as such become easy targets for hackers. This could cause the company or the individual to lose a lot of money. It is not just the developers who are affected, end users who end up on vulnerable websites may get exposed to XSS attack which could compromise their system or an unsecured configuration of database system could lead to a potential data leak and hence the password of every registered user on the website is compromised, users who use the same password on multiple websites are affected the most. The motivation for this paper comes from the fact that there is an overwhelming number of vulnerabilities in any application under development and every developer, experienced or not needs a starting point to patch these vulnerabilities that might have occurred in their application, this research provides the most common vulnerabilities which should be taken care of in any application and thus provide the much-needed starting point for developers. The objective of this paper is to design and develop a secure web application according to Open Web Application Security Project (OWASP) guidelines. This paper highlights the mitigation of vulnerabilities in the web application using configuration changes, coding and applying patches. The vulnerabilities SQL injection, Broken authentication, Sensitive data exposure, Broken Access Control, and XML external entities discussed in this paper are listed under the OWASP top 10 vulnerabilities. The security of the web application is tested and proved to have defense mechanism implemented for the mentioned vulnerabilities.**

***Keywords*—Vulnerabilities, OWASP, SQL injection, Session Management, Broken authentication, Password Hashing, Sensitive data exposure, and Server verification.**

I. INTRODUCTION

A Web application (Web app) is an application program that is stored on a remote server and delivered over the Internet through a browser interface. A web application generally consists of three main parts –

- Front – End,
- Back – End and
- Database.

There are many providers for the above three components. Some examples are –

- Front – End - Angular, ReactJS, Vue.js, etc.
- Back – End - NodeJS, php, Laravel, etc.
- Database – MySQL, MongoDB, Cassandra, Redis, etc.

For this paper, the front-end has been made using HTML with ejs, for back-end NodeJS has been used and for the database MySQL has been used.

A. The Frontend:

The front end is used to display the main content of the webpage, it is usually the only thing that the client sees once the site is visited. If the website is static then only the front end is required.

HTML is used to form the layout of the whole page, CSS is used to give styling to the page, and JavaScript is used to give logic to the page. Websites having only the frontend are prone to Cross-Site Scripting (XSS) injection attacks.

Advanced technologies like Angular and ReactJS have a built-in mechanism to prevent XSS injection however while building a website using the primitive technologies various steps have to be taken care of to successfully prevent XSS injection.

B. The Backend:

Backend is required for dynamic website, it is usually responsible for communicating with the database and providing data to the front end to display.

Any sensitive information is usually kept in the backend and never sent to the frontend because the client can see the frontend data however, it is extremely hard to access the data at the backend. The frontend languages can only run in the browser however the backend languages can be used to communicate with the system they are compiled on.

C. The Database:

The database is the place to store any data which has to be retained and should not be accessible by the client. There are mainly two types of database – SQL database and NoSQL database.

1) SQL database:
These types of databases are used when the data to be stored is structured and the structure is not expected to change.

2) NoSQL database:
When the data has no structure and could vary from user to user then a NoSQL database is used.

II. OBJECTIVE

The main objective of this paper is to provide methods to build a secure web application using NodeJS. This paper focuses on the various vulnerabilities and the methods to secure the web application from these vulnerabilities. This paper also provides the code that can be used to secure the web application.

This paper also explains the levels of security that a developer should maintain to ensure that the user data is

2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS) | 978-1-6654-1272-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICICCS51141.2021.9432179

always protected. These levels are discussed first in this paper and then the methods to make the web application secure follow.

III. PROPOSED IDEA

The task at hand revolves around eliminating the OWASP vulnerabilities for a self-tailored site that consists of most of the vulnerabilities highlighted by OWASP has been used for testing. In the given figure (Fig. 1) of the proposed idea, the "OWASP Vulnerabilities" column highlights the vulnerabilities that have been considered for this paper. The "Vulnerability Location" column points out the features in the web application where there is a possibility of an attack and which should be implemented keeping in mind the possible attacks. The column "Vulnerability Prevention" describes the steps that can be taken for the elimination of the vulnerability from the web application.

The website is an MCQ test-taking website that generates a certificate of either participation or merit, based upon the result of the test. The objective of this paper is to use this website to show how to eliminate all the possible vulnerabilities. SQL injection attack is eliminated during the login and registration by hashing the password and sanitizing the code.

Sensitive data exposure is prevented by not sending any sensitive data to the client-side and using only the server to verify the details as well as the answers. Also, the database is hashed using SHA256 with salt to provide maximum security for the registered users.

Broken Authentication is prevented by using sessions to store the login details of the currently active users. This redirects the users to the home page if they access a domain they are not supposed to. Sessions also help to log-out a particular user successfully. Also, XML External Entities can be prevented by using a JavaScipt Object Notation (JSON) Object in which malicious codes could not be passed.

Broken access control is prevented by password-protecting pages that are not supposed to be visited by the client.

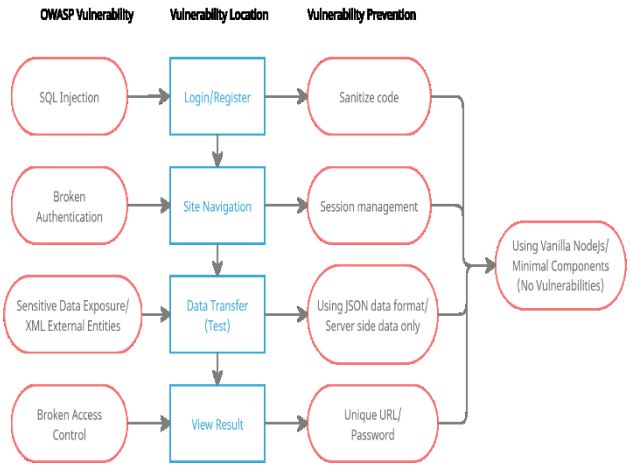

Fig. 1 – The proposed idea.

IV. RELATED WORKS

As the technology becomes better and complex with the increase in vulnerability for a cyber threat. Different testing techniques help to find a loophole in the security. Following a standard procedure and defined policy can reduce these loopholes. To mitigate the threats, it is necessary to install security patches. [1] Patel. K in his paper includes a survey on the current vulnerabilities and the detection of those vulnerabilities.

Network threats often come from multiple sources and affect a variety of domains. Collaborative sharing and analysis of Cyber Threat Information (CTI) can greatly improve the prediction and prevention of cyber-attacks. [2] However, CTI data containing sensitive and confidential information can cause privacy exposure and disclosure of security risks, which will deter organizations from sharing their CTI data. This concern can be handled by creating API Gateway, which provides a bridge between end-users and their data sources. Through this, users can select which data is sharable with privacy-preserving means.

Cross-site scripting (XSS) refers to an unintentional client-side code injection attack. It makes use of un-validated or un-encoded user input and returns the malicious output. [3] In type 1 XSS, malicious code or script is embedded in a Web request. It is usually done through an email that encourages the user to click on a provided malicious link or a website with a malicious link. This can be greatly avoided by not following the malicious links. This vulnerability is one of the most widespread security problems for web applications. Various approaches to defending against attacks that use XSS vulnerabilities are available today but no single approach solves all the loopholes. [4] An efficient approach discussed by I. Yusof and A. K. Pathan to prevent persistent XSS attacks is by applying the pattern filtering method.

XSS attack takes the user to a webpage designed to steal user's sessions and cookies. Nearly 68 percent of websites are vulnerable to XSS attacks. [5] G. Habibi and N. Surantha suggest equipment of machine learning methods namely Support Vector Machine (SVM), K-Nearest Neighbour (KNN), and Naïve Bayes. Machine learning algorithm equipped with the n-gram method improve the detection performance of XSS attacks.

SQL injection could be easily implemented using some set of special characters. The PHP framework implemented a method of Magic Quotes that adds '\' in front of the special characters such as \,' and " which is transformed to \\, \' and \". [6]

Penetration testing is an important part of any application deployment. Penetration testing can help to identify potential vulnerabilities and insight on how those vulnerabilities could be mitigated. [7] A. Goutam and V. Tiwari in their paper have come up with a framework to test the vulnerability of a financial application.

SQL injection is a very basic yet extremely powerful method of gaining access to any system. In applications where the user details are verified during the login process are highly susceptible to this form of attack. [8] P. N. Joshi, N. Ravishankar, M. B. Raju, and N. C. Ravi developed a secure system for authentication access and applied SQL injection attacks to check the security.

M. Dua and H. Singh [9] discuss the various attacks possible attacks on a website like SQL injection and XSS attacks. Their idea uses a XAMPP server for client and server communication.

R. A. Katole, S. S. Sherekar and V. M. Thakare [10] discuss the importance of SQL injection attacks in modern-day websites. They have also proposed an idea to detect SQL injection attacks by removing the parameter values of the SQL query.

Session management is an important area to consider while developing any web application. Sessions that are not handled properly could lead to the leakage of information. R. Lukanta, Y. Asnar and A. I. Kistijantoro [11] developed a tool to detect session vulnerability, this tool adds more features on top of currently existing tools like Nikto.

Registering different users without checking the validity of every user could easily lead to a serious vulnerability that could later be used to perform a DOS attack on the system. A good way of checking the validity of the emails and the user is through the Time-Based One-Time Password proposed by H. Seta, T. Wati, and I. C. Kusuma [12]. This way a token will be generated which could validate the user and hence prevent DOS attacks on a smaller scale. The authors decided to use hashing for an extra layer of security. This paper however uses session management to keep the OTP confidential. Therefore, there is no need to apply hash on an OTP which could potentially slow down the process.

Just the knowledge of different vulnerabilities and the ways to mitigate them does not ensure that one could develop a secure system. A. Anis, M. Zulkernine, S. Iqbal, C. Liem and, C. Chambers [13] in their paper discuss the various secure coding practices which could help developers develop secure code on the client-side. The approach is built on JavaScript which is the language used by the majority of web applications.

K. Vijayalakshmi and A. A. Leema [14] point out that most of the XSS prevention mechanism use simple filtering that cleans the malicious code, however, this filtering process does not work for some cases and hence they have proposed an approach in their paper which takes care of few of the edge cases where simple XSS might fail.

Different technologies used for developing web applications might have different types of vulnerabilities that should be taken care of. Therefore, the implementation of detection and prevention mechanisms vary from language to language. H. AL-Amro and E. El-Qawasmeh [15] discuss various ways to detect vulnerabilities on websites made with ASP.NET.

When a new vulnerability is detected, it has to be recorded in Common Vulnerabilities and Exposures (CVE) and it also has to be scored by Common Vulnerabilities Scoring System (CVSS), however, this takes a lot of time. D. Iorga, D. Corlătescu, O. Grigorescu, C. Săndescu, M. Dascălu and R. Rughiniș [16] proposed the idea of using machine learning models to detect vulnerability in news website. Various models were trained and their performances were recorded.

There are various ways to detect XSS attacks, it quickly becomes a mammoth task to choose which would work the best for the given website. M. Liu, B. Zhang, W. Chen and X. Zhang [17] in their paper have analyzed various XSS detection mechanisms, classified them into 3 different categories, and performed a survey on their performance.

A vulnerable website could indirectly affect the end-users as well. A phishing website is created when a vulnerable website is scanned and its counterfeit website is hosted without the knowledge of the server owner. B. Wardman, G. Shukla, and G. Warner [18] have discussed the common vulnerabilities which allow such phishing sites to show up, and also their idea informs the owner if their website has been compromised.

It's not just the website that could host an enormous number of vulnerabilities, some technology used inside the website could also be the cause for a breach. For example, a website using a QR code to send documents. The website may not be vulnerable however QR code in itself has several vulnerabilities which might not be known to the developer. A. Averin and N. Zyulyarkina [19] discuss the threats in using QR codes in blockchain applications.

F. Yu and Y. Tung [20] have developed an online service where developers could detect view and patch the various vulnerabilities on their websites. It is done using static string analysis.

J. Zhao and R. Gong [21] and H. AL-Amro and E. El-Qawasmeh [22] have discussed SQL injection, Cross-Site Scripting session hijacking, and other vulnerabilities in PHP and ASP.NET respectively. They have shown the ways to fix the vulnerability but not the consequences of not fixing the vulnerability.

V. LEVELS OF USER AUTHENTICATION AND SECURING USER DATA

A. Level - 1

Usage of username and password to login to a specific user's portal. This is essential so that no other user can intrude on the privacy of any other user by accessing their profile.

B. Level – 2

Authenticating the user by sending a mail with a One-Time-Password (OTP) to their email ID and verifying the OTP at the time of registration. This will validate that the user is a valid one.

C. Level – 3

The user's password should be hashed using hashing algorithms such as SHA-256. This ensures that in the case of a data breach the user data is safe from the attacker, it is safe because decryption of a string hashed with SHA 256 using a computer takes approximately 4 trillion years.

D. Level – 4

To ensure that the user's password is safe the password can be salted (salting adds some extra string to the user's password and then hashes it). This results in an even longer string and will take an even longer time to decrypt.

E. Level – 5

Enabling Two-Factor Authentication. Letting the user set up the number and verify it. This allows the user to log-in only if they provide the correct OTP sent to their mobile number at the time of login.

VI. ARCHITECTURE

A. Insecure Web Application (Environment and Detection) -

The initial environment for the research was taken to be an insecure web application having no layer of security. This app was tested for OWASP vulnerabilities. Following are the conclusions drawn from the testing –

1) SQL Injection –

On entering the SQL commands to drop the user's table in the login form, the table was dropped from the database. This occurred because the form was not able to process whether this was a normal insertion in the table or a command to do some manipulation.

2) Brute Force Login Attack –

A vulnerability testing tool such as Burp-Suit can implement multiple login attempts by using techniques such as permutation to predict the correct login credentials. The tool checks for the "route" to which the individual credentials lead to.

The credentials leading to pages such as "Home", etc. are the correct login credentials and could be used to access the personal information of the user. On implementing this, the tool did give the credentials that were leading to the "Home" route.

Also, a higher value was given for those credentials whose password might be right.

3) Password Sniffing –

A password that is not encrypted before storing has a high chance of getting sniffed and used by hackers. The hackers can use SQL injection methods to read the passwords of users.

The website did not store the passwords in the encrypted format in the database and could be easily used by the hackers to extract sensitive information related to the users.

4) External XML Entities –

XML is used to transfer data between client and the server. For example, the questions are sent in XML format to display to the users. For easy management of XML data from the client-side vulnerable XML parsers are used as external libraries on the server-side. This causes the system to become insecure.

5) Broken Authentication –

If the user logs out i.e., clicks the "Log Out" button, the user should be taken to the root page of the website and under no circumstances the user should be able to view the contents of his profile without logging in.

The website did take the user to the root page i.e., the login page of this website, but once the back button was pressed, the user could easily get back to his profile and view the contents as if they were logged in.

B. Secure Web Application (Prevention) –

After the initial tests, the web application was made secure using the steps that follow this section but the results that were brought out are as follows –

1) SQL Injection –

The web application no longer processed any type of SQL commands and just pointed out that the credentials were wrong.

2) Brute Force Login Attack –

Providing an additional Completely Automated Public Turing test to tell Computers and Humans Apart (CAPTCHA) field increased the number of permutations one must need to have, also this CAPTCHA code refreshes on a new login attempt, so there is less probability that the attacking tool could guess the correct credentials.

This method did solve the problem for the Brute Force Login Attack for the web application.

3) Password Sniffing –

The passwords were encrypted using Data Encryption Standard (DES) hashing algorithm. In this case even if one gets successful at extracting the encrypted passwords from the database, it would take years for them to decrypt one password.

The encryption was successfully done for the passwords for the considered web application and the login was also successful after the encryption.

4) External XML Entities –

All XML transfers between the client and the server are removed and replaced with JSON transfer. All the XML parser dependencies from the server file code have been removed. Changed the HTTP request header to JSON type. (Content-type/JSON).

For backend languages that require additional libraries to process Json their dependencies should be added. Changed the client side code to receive and send JSON objects. JavaScript has a built-in JSON parser.

5) Broken Authentication –

This issue was tackled by implementing sessions in the web application. As soon as a user clicks the "Log Out" button the session corresponding to that individual user is destroyed. So in this scenario, unless the session is started for the user using the correct credentials the user cannot return to their profile. This prevention technique was successfully implemented for the web application.

VII. METHODS TO SECURE THE WEB APPLICATION

A. Architecture Diagram –

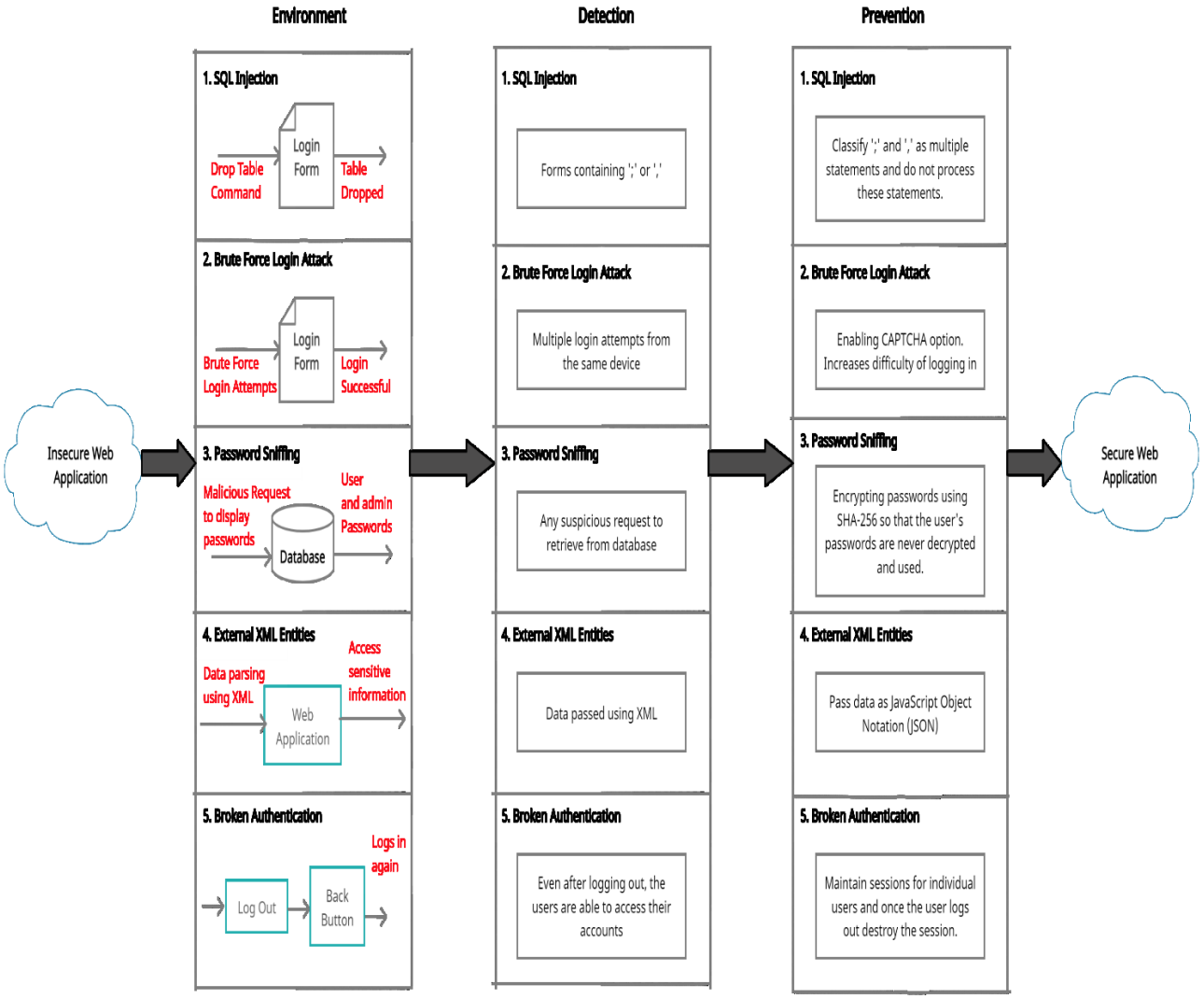

Fig. 2 – The steps that are followed to develop a secure web application.

1) Environment –

The "Environment" in the above architecture diagram shows the insecure web application.

2) Detection –

The "Detection" column shows the hints and errors one can check to ensure if the website is vulnerable to attacks.

3) Prevention –

The "Prevention" column shows the steps one can follow to make the web application secure.

VIII. IMPLEMENTATION

A. *SQL Injection –*

Using SQL injection attacker can gain access to the entire database using reverse shell connection. It is an extremely dangerous attack that should be taken care of in any web application.

The attacker sends a malformed query in the input text, the query is formed such that the query results to true even if the password or username is wrong. This way the attacker gets access to the database.[1][3]

The major reason why it occurs is that the input from the user is not parsed before using the input in the search query.[7]

To avoid SQL injection 2 things can be done in NodeJs –

1) Instead of appending the queries with the query string, use a binding function to bind it to the string.[8]

2) Always turn off multiple SQL statements in the SQL connection, this way the attacker cannot use the multiple statements to evaluate an expression to true. The attacker can gain root access using SQL injection vulnerability.

- Example –
  a. Multiple Statements –

i. Malformed Code –

```
var mysql = require("mysql");
var connection = mysql.createConnection({
host:"localhost",user:"root",password:"root",
database:"vlab_check",multipleStatements: true
});
module.exports = connection;
```

If this statement is used, then the attacker will be able to send queries through the forms which will be processed by the backend.

ii. Corrected Code –

```
var mysql = require("mysql");
var connection = mysql.createConnection({ host:
"host",
user: "username", password:
"password", database:
"database_name",
multipleStatements: false
});
module.exports = connection;
```

b. Binding the query-

i. Malformed Code –

```
sql.query("select * from users where
rollno='"+rollno+"' and
password='"+password+"'",function(err,result){
if(err) { console.log(err); } else {
if(result.length != 0)
{
        req.session.userId = rollno;
        res.redirect("/home");
    }
  }
});
```

If this statement is used then the queries that are entered in the login forms will be placed in the '?' as it is. This format of insertion may lead to some serious issues like dropping of table etc.

ii. Corrected Code-

```
sql.query("select * from users where
  rollno=?",[rollno],function(err,result){
    if(err) { console.log(err); }
     else {
      if(result.length != 0)
      {
         if(x == result[0].password)
         {
            req.session.userId = rollno;
            res.redirect("/home");
         }
    }
  }
  }
```

B. *Broken Authentication –*

It usually occurs when the users can brute force and find the password, or when the user can go back to the login page after logging in.

Using this the attacker can access the data of another user. Or in shared computers when a person logs out and still another person using the same PC can access the account of the other person due to poor session management. Usually, intruder attacks like Battering Ram and cluster bomb attacks are used to exploit broken authentication.

Avoiding broken authentication –

1) To prevent a brute force attack on the log-in page, the easiest and safest method is to use CAPTCHA. Since CAPTCHA cannot be filled by the automation tool, user intervention is required and the automation of attack cannot be done.

2) To prevent automation tools from finding other passwords once a password has already been found, use sessions and cookies to avoid going back to the login page again after log in has been done and the user did not log-out. This way the user has to manually run the attack every time a password has been found.[9]

Since HTTP requests are stateless, session management is important to identify different users and also to identify which user has logged out and a password should be entered if the user logs in again.

The following code snippet is a middle-ware that is, it is run after the request from the user and before the response from the server, hence it becomes the perfect place to check for a session. If the user has

logged in then the user id of the session should be set, therefore if the user tries to access the route without logging in with an undefined user id, this middleware will redirect the user to the login page.

Similarly, if the user is giving a test and goes back to the log-in page due to an internet issue, then after logging in the user is redirected to the test from where he left.

- Example –
  Cookies and Sessions –

```
var middleware = function(){
// In the login page
return function(req,res,next){
if(req.session.userId === undefined){
// checks if user has already logged in
next();
}
else if(req.session.test){
// checks if the user is giving test
res.redirect("/test/review/"
+req.session.subject);
}
else{ res.redirect("/home") }
}
}
```

C. Sensitive Data Exposure –

This attack varies from application to application as well as from person to person. It depends upon the application and the person to classify a piece of information as sensitive.

This vulnerability is extremely important to fix in high-profile web pages like banking sites. If the account details of one person are exposed to the attacker this could be a huge problem for both the bank and the victim.

Sensitive data exposure could be done without one's knowledge, as data that does not seem as sensitive to a person may be seen sensitive in the eyes of the other person.

Ways to handle sensitive data –

1) Always use the server to handle sensitive data. Like in our case the answers to the questions are never sent to the client-side and all the verification is always done at the server-side, this is because the data on the client-side can easily be viewed by the client, this is not the case with server-side code.

2) Data related to passwords must always be hashed and stored, this is because in the future if there is any breach the passwords of the users are safe as it can be extremely difficult and time-consuming to crack encryptions like SHA256 with salt.

- Example –
  Hashing –

i. Malformed Code –

```
var rollno = req.body.regno.toUpperCase();
var password = req.body.password;
```

Password is stored in the database the way user types without any encryption.

iii. Corrected Code –

```
var crypto = require("crypto-js");
password = crypto.SHA256(password).toString();
```

Password is changed to the updated encrypted password and store in the database. Password is the input gotten from the user.

D. Server Verification –

To client –

Test of subject, question index, Number of questions, the question with 4 options, the button status (flagged, answered, unvisited), the time.

As it can be seen the answer is not sent to the client, only the question and option.

- Example –

```
res.render("test.ejs",{subject: currentUser.subject,
questionIndex:
currentUser.questionIndex,numOfQuestions:
NUM_OF_QUESTIONS,
question:currentUser.questionMatrix
[currentUser.questionIndex],
buttonStatus: currentUser.buttonStatus,
timeValue: currentUser.time});
```

E. XML External Entities –

This vulnerability is mainly due to the data exchange using XML. XML External Entities is the vulnerability that is caused when a malicious XML script that contains a reference to an external entity is parsed by the web application and runs the malformed code that is inside that script. XML is used as a data object for sending data through API calls. If malicious data is sent via the API call an attack can be easily executed and this attack may result in denial of service, disclosure of confidential data, server-side request forgery, port scanning, etc. A simple approach to eliminate this vulnerability is to change the data object that is used to send the data via the API calls. The web application being discussed uses JavaScript Object Notation (JSON) object for transferring data via the API calls. This data object eliminates the possibility of an XML External Entities attack. This attack occurs when XML input containing a reference to an external entity is processed by a weakly configured XML parser.[4][6]

This is perhaps the easiest vulnerability to fix. Just use JSON (JavaScript Object Notation) to pass the data between client and server, in single-page applications using REST Application Programming Interface (API).

Most the major backend services now provide native support for JSON data transfer.

- Example –

res.render("home.ejs",{isLoggedIn: isLoggedIn});

Rendering the home page, with login information sent as a JSON object to the ejs viewer.

In this example, isLoggedIn on the left side is the key and isLoggedIn variable on the right is the value. It is a common convention to name both the key-value pairs as the same variable name while sending data to ejs.

IX. RESULTS

A. SQL –

NodeJS is a relatively new backend language therefore automated vulnerability scanners don't work well with most of the NodeJS applications. Manual testing of vulnerabilities is one of the ways to validate the security of the application.

```
mysql> select rollno from users;
+-----------+
| rollno    |
+-----------+
| 18BCE1001 |
| 18BCE1000 |
| 18BCE1025 |
| 18BCE1033 |
+-----------+
4 rows in set (0.00 sec)
```

Fig. 3 – Users in the database

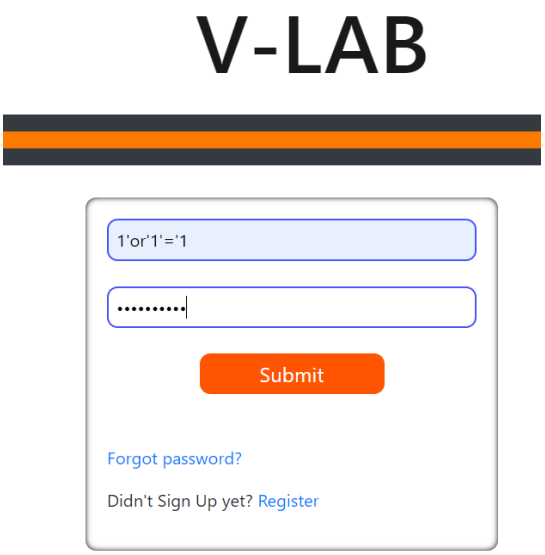

Fig. 4 – Standard SQL injection

1'or'1'='1, is a standard SQL injection statement.
The SQL query to retrieve a user based on username and password is
Select * from user_table where username = 'username' and password = 'password'
The underlined variables are replaced by the text user types, in the case of SQL injection, both username and password are: 1'or'1'='1.
Therefore, the statement now becomes,
Select * from user_table where username = '1'or'1'='1' and password = '1'or'1'='1'
'1' = '1' is always true and hence the whole query valid. Therefore, all the details present in the database are listed.

```
mysql> Select rollno from users where rollno = '1'or'1'='1' and password = '1'or'1'='1';
+-----------+
| rollno    |
+-----------+
| 18BCE1000 |
| 18BCE1001 |
| 18BCE1025 |
| 18BCE1033 |
+-----------+
4 rows in set (0.00 sec)
```

Fig. 5 – Execution of malicious SQL query

Hence the attacker can gain entry into the site without being a legitimate user.
However, when the security measures are applied in the code such as binding the query instead of writing the variable names appended with quotation marks the resulting query becomes:
Select * from user_table where username = 1'or'1'='1 and password = 1'or'1'='1
When this statement is entered as a query the result is an error therefore the attacker does not gain access to the site.

```
mysql> Select * from user_table where username = 1'or'1'='1 and password =
 1'or'1'='1;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual
 that corresponds to your MySQL server version for the right syntax to use
 near ''or'1'='1 and password = 1'or'1'='1' at line 1
mysql>
```

Fig. 6 – Handling malicious query

Therefore, SQL injection is handled correctly by binding query.

B. Broken Authentication –

HTTP is a stateless protocol, that is there is no connection between the current request and previous request, all of the requests are anonymous. Therefore, to identify different users the developers usually store temporary data in the session storage of the browser. Session storage is deleted once the browser is closed or after the time limit for the data is elapsed.

In practice a misconfigured session management could result in the following scenario, A user logs into the site, then logs out and does not close the browser, another user could just press the back button on the browser and the previous user's page is shown to the attacker, this happens because once the previous user logs out the session is not destroyed, and once the back button is pressed the session is restored.

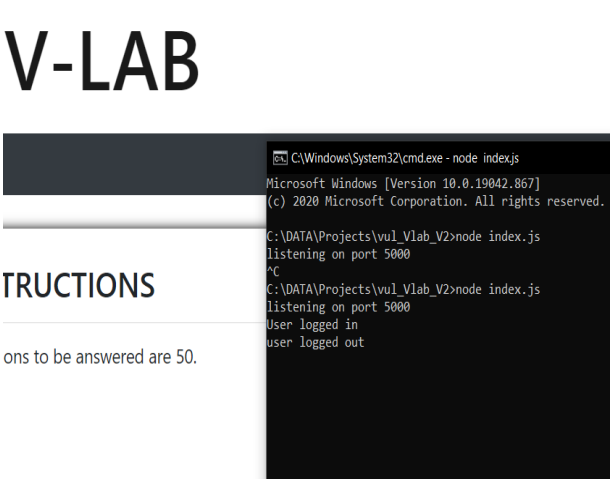

Fig. 7 – Misconfigured session

It can be seen in the logs that the user logged in, then logged out, however upon pressing the back button the user is once again inside the website and it does not show up in the logs because browsers cache the previously visited pages for faster retrieval of visited pages. However, this poses a security threat as can be seen in this example. The way to patch this is to tell the browser to load the backend script of the page again and not display the page from the cache.

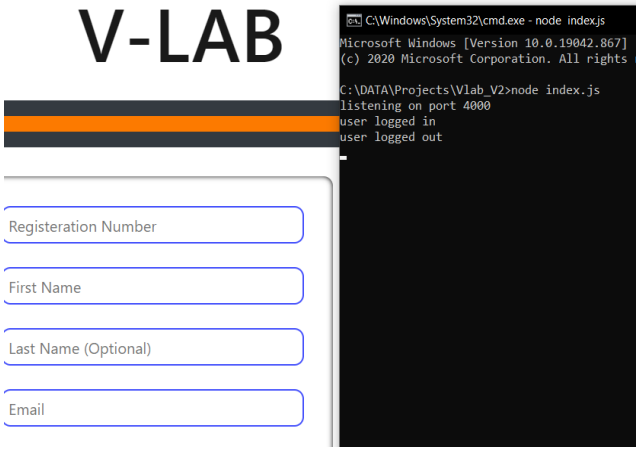

Fig. 8 – Session configuration

After fixing the vulnerability it can be seen that when the back button is pressed the user is taken to the registration page instead of the home page of the previous user.

X. NOVELTY

This is research is done using NodeJS as a backend language. NodeJS uses JavaScript to handle the backend process of the web application. It is a relatively new technology that is gaining popularity very quickly however there aren't many works that focus on the security aspect while using NodeJS. This paper introduces the common vulnerabilities in a language-independent form and the patch for those vulnerabilities using NodeJS. There are numerous papers and articles which provide implementation detail of vulnerabilities using PHP for instance [21] provides the vulnerability patch details of SQL injection, Cross-Site Scripting and various other vulnerabilities using PHP. [22] provides the patch details in ASP.NET applications. However, not many papers provide the details in NodeJS. Therefore, the novelty of the research work lies in the implementation of security features using NodeJS.

XI. CONCLUSION

The purpose of this research was to find all the weak points in a web application corresponding to the OWASP guidelines. Based on the analysis, it can be concluded that a simple web application with no security is highly prone to attacks and can be easily taken down. Basic SQL queries for deleting the database will be executed with no checks in a simple web application, and this alone is enough to bring down the web application. After the implementation of the security guidelines provided by OWASP, the web application taken for consideration showed no negative results for the attacks that were performed on the web application earlier. The attacks such as SQL injection, XSS did not do any negative effect on the application. The limitation of this research is that it is implemented for the web application that had backend as NodeJs. Other backend service providers such as php, Java, etc. may not be able to deal with the implementation idea provided in this research paper. Another limitation of this research paper was that the testing of the mentioned vulnerabilities was performed with some specific tools and with certain levels of testing. The web application may behave differently if the levels of the testing are changed.

REFERENCES

[1] K. Patel, "A Survey on Vulnerability Assessment & Penetration Testing for Secure Communication," 2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI), Tirunelveli, India, 2019.

[2] W. Fan et al., "Enabling Privacy-Preserving Sharing of Cyber Threat Information in the Cloud," 2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/ 2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom), Paris, France, 2019.

[3] S. N. Bukhari, M. Ahmad Dar and U. Iqbal, "Reducing attack surface corresponding to Type 1 cross-site scripting attacks using secure development life cycle practices," 2018 Fourth International Conference on Advances in Electrical, Electronics, Information, Communication and Bio-Informatics (AEEICB), Chennai, India, 2018.

[4] I. Yusof and A. K. Pathan, "Preventing persistent Cross-Site Scripting (XSS) attack by applying pattern filtering approach," The 5th International Conference on Information and Communication Technology for The Muslim World (ICT4M), Kuching, Malaysia, 2014.

[5] G. Habibi and N. Surantha, "XSS Attack Detection With Machine Learning and n-Gram Methods," 2020 International Conference on Information Management and Technology (ICIMTech), Bandung, Indonesia, 2020.

[6] J. Kim, "Injection Attack Detection Using the Removal of SQL Query Attribute Values," 2011 International Conference on Information Science and Applications, Jeju, Korea (South), 2011.

[7] A. Goutam and V. Tiwari, "Vulnerability Assessment and Penetration Testing to Enhance the Security of Web Application," 2019 4th International Conference on Information Systems and Computer Networks (ISCON), Mathura, India, 2019.

[8] P. N. Joshi, N. Ravishankar, M. B. Raju and N. C. Ravi, "Contemplating Security of Http From SQL Injection and Cross Script," 2017 IEEE International Conference on Computational

Intelligence and Computing Research (ICCIC), Coimbatore, India, 2017.

[9] M. Dua and H. Singh, "Detection prevention of website vulnerabilities: Current scenario and future trends," 2017 2nd International Conference on Communication and Electronics Systems (ICCES), Coimbatore, India, 2017.

[10] R. A. Katole, S. S. Sherekar and V. M. Thakare, "Detection of SQL injection attacks by removing the parameter values of SQL query," 2018 2nd International Conference on Inventive Systems and Control (ICISC), Coimbatore, India, 2018.

[11] R. Lukanta, Y. Asnar and A. I. Kistijantoro, "A vulnerability scanning tool for session management vulnerabilities," 2014 International Conference on Data and Software Engineering (ICODSE), Bandung, Indonesia, 2014.

[12] H. Seta, T. Wati and I. C. Kusuma, "Implement Time Based One Time Password and Secure Hash Algorithm 1 for Security of Website Login Authentication," 2019 International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS), Jakarta, Indonesia, 2019.

[13] A. Anis, M. Zulkernine, S. Iqbal, C. Liem and C. Chambers, "Securing Web Applications with Secure Coding Practices and Integrity Verification," 2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech), Athens, Greece, 2018.

[14] K. Vijayalakshmi and A. A. Leema, "Extenuating web vulnerability with a detection and protection mechanism for a secure web access," 2017 Fourth International Conference on Signal Processing, Communication and Networking (ICSCN), Chennai, India, 2017.

[15] H. AL-Amro and E. El-Qawasmeh, "Discovering security vulnerabilities and leaks in ASP.NET websites," Proceedings Title: 2012 International Conference on Cyber Security, Cyber Warfare and Digital Forensic (CyberSec), Kuala Lumpur, Malaysia, 2012.

[16] D. Iorga, D. Corlătescu, O. Grigorescu, C. Săndescu, M. Dascălu and R. Rughiniș, "Early Detection of Vulnerabilities from News Websites using Machine Learning Models," 2020 19th RoEduNet Conference: Networking in Education and Research (RoEduNet), Bucharest, Romania, 2020.

[17] M. Liu, B. Zhang, W. Chen and X. Zhang, "A Survey of Exploitation and Detection Methods of XSS Vulnerabilities," in IEEE Access, vol. 7.

[18] B. Wardman, G. Shukla and G. Warner, "Identifying vulnerable websites by analysis of common strings in phishing URLs," 2009 eCrime Researchers Summit, Tacoma, WA, USA, 2009.

[19] A. Averin and N. Zyulyarkina, "Malicious Qr-Code Threats and Vulnerability of Blockchain," 2020 Global Smart Industry Conference (GloSIC), Chelyabinsk, Russia, 2020.

[20] F. Yu and Y. Tung, "Patcher: An Online Service for Detecting, Viewing and Patching Web Application Vulnerabilities," 2014 47th Hawaii International Conference on System Sciences, Waikoloa, HI, USA, 2014.

[21] J. Zhao and R. Gong, "A New Framework of Security Vulnerabilities Detection in PHP Web Application," *2015 9th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, Santa Catarina, Brazil, 2015.

[22] H. AL-Amro and E. El-Qawasmeh, "Discovering security vulnerabilities and leaks in ASP.NET websites," *Proceedings Title: 2012 International Conference on Cyber Security, Cyber Warfare and Digital Forensic (CyberSec)*, Kuala Lumpur, Malaysia, 2012.