

DESIGN OF A SECURITY SYSTEM FOR WEB ATTACKS

THESIS

Submitted in fulfillment of the requirement of the degree of

DOCTOR OF PHILOSOPHY

to

YMCA UNIVERSITY OF SCIENCE & TECHNOLOGY

by

BHARTI NAGPAL

Registration No. YMCAUST/PhD03/2012

Under the Supervision of

Dr. NARESH CHAUHAN

PROFESSOR(YMCAUST)

Dr. NANHAY SINGH

ASSOCIATE PROFESSOR(AIACT&R)



**Department of Computer Engineering
Faculty of Engineering and Technology
YMCA University of Science & Technology
Sector-6, Mathura Road, Faridabad, Haryana, India**

FEBRUARY, 2018

Dedicated

to

My daughter Lavanaya

DECLARATION

I hereby declare that this thesis entitled “**DESIGN OF A SECURITY SYSTEM FOR WEB ATTACKS**” being submitted in fulfillment of requirement for the award of Degree of Doctor of Philosophy in the **DEPARTMENT OF COMPUTER ENGINEERING** under Faculty of **ENGINEERING & TECHNOLOGY** of YMCA University of Science and Technology, Faridabad, during the academic year March 2013 to February 2018, is a bonafide record of my original work carried out under the guidance and supervision of **DR. NARESH CHAUHAN, PROFESSOR, DEPARTMENT OF COMPUTER ENGINEERING, YMCA UNIVERSITY OF SCIENCE & TECHNOLOGY, FARIDABAD** and **DR. NANHAY SINGH, ASSOCIATE PROFESSOR, DEPARTMENT OF COMPUTER ENGINEERING, AMBEDKAR INSTITUTE OF ADVANCED COMMUNICATION TECHNOLOGY & RESEARCH, DELHI** and has not been presented elsewhere.

I further declare that the thesis does not contain any part of any work which has been submitted for the award of any degree either in this university or in any other university.

(BHARTI NAGPAL)

Registration No: YMCAUST/Ph03/2012

CERTIFICATE

This is to certify that this thesis entitled “**DESIGN OF A SECURITY SYSTEM FOR WEB ATTACKS**” by **BHARTI NAGPAL** submitted in fulfillment of the requirements for the award of Degree of Doctor of Philosophy in **DEPARTMENT OF COMPUTER ENGINEERING** under Faculty of **ENGINEERING & TECHNOLOGY** of YMCA University of Science & Technology Faridabad, during the academic year March 2013 to February 2018 is a bonafide record of work carried out under our guidance and supervision.

We further declare that to the best of our knowledge, the thesis does not contain any part of any work which has been submitted for the award of any degree either in this university or in any other university.

DR. NARESH CHAUHAN

PROFESSOR

Department of Computer Engineering
Faculty of Engineering and Technology
YMCA University of Science & Technology,
Faridabad

DR. NANHAY SINGH

ASSOCIATE PROFESSOR

Department of Computer Engineering
Ambedkar Institute of Advanced
Communication Technology &
Research, Delhi

Dated:

The Ph.D viva-voce examination of Research Scholar Bharti Nagpal(YMCAUST/Ph03/2012)

has been held on.....

(Signature of Supervisors)

(Signature of Chairman)

(Signature of External Examiner)

ACKNOWLEDGEMENT

I am thankful to God for all of His Blessings

I would like to express my sincere gratitude to my thesis supervisors, **Dr. Naresh Chauhan**, Professor, Department of Computer Engineering, YMCA University of Science & Technology, Faridabad and **Dr. Nanhay Singh**, Associate Professor, Department of Computer Engineering, Ambedkar Institute of Advanced Communication Technology & Research, for their continuous guidance, valuable advice, constructive criticism and helpful suggestions. I am very grateful to them for their continual encouragement and motivation. I greatly value their timely and valuable advices. Their suggestions increased my cognitive awareness and helped considerably in the fruition of my objective.

I am indebted to **Dr. Komal Kumar Bhatia**, Professor and Chairman, Department of Computer Engineering, YMCAUST for his insightful comments and administrative help at various occasions. I am indebted to **Dr. C. K. Nagpal**, Professor and **Dr. Atul Mishra**, Associate Professor, Department of Computer Engineering, YMCAUST for their valuable suggestions. I would also like to thank my DRC members, **Dr. Neelam Duhan**, **Dr. Komal Kumar Bhatia**, **Dr. Manjeet Singh** and **Dr. Ashutosh Dixit** for stimulating questions and invaluable feedback. I expand my thanks to all the faculty members of Computer Engineering Department of YMCAUST for their support and cooperation.

I would like to thank my parents, my sister and my brother for their guidance, love and encouragement. My sincere thanks to my husband for providing the constant encouragement and unconditional moral support to enable me to come up to this level in my life. My special thanks to my daughter for the encouragement, the priceless love and support.

Thanks to all of you!

(BHARTI NAGPAL)

ABSTRACT

Information is an important resource and achieving a proper security level can be viewed as basic keeping in mind the end goal to keep up a focused edge. The advancement of the web has established a framework for the improvement and use of new classification of system of information technology working on the web. The use of internet has been tremendously grown for accomplishing important task, such as online shopping, e-banking, e-reservation, e-governance etc.

Nowadays, the web has turned out to be extremely fundamental needs of the general public and in like manner there is an expanding interest to comprehend the exercises, offices, and objectives of web clients. WWW has developed quickly into a flexible stage for a wide range of computation, dynamically picking up support for information passage, client side scripting, and application-particular system dialogues.

With the quickly developing technology, the simplicity of accessing through web applications has changed the conventional perspective of an organization completely. Hacking of webpages of web applications keep on gaining notoriety as hackers are exposing vulnerabilities over all geographies and crosswise over different sorts of web technologies. Hackers are always exploring different avenues regarding an extensive variety of assaulting strategies to trade off sites and hack delicate information, for example, MasterCard number, social security number and other individual data.

Security has turned into a noteworthy aspect to the internet world as internet age is rising day by day. Security of the web application is important of the fact that now every activity like sharing, communication, sharing the assets, e-administering, online managing a bank account, online business, social communication, payment of different utilities bills etc. done on the internet.

Web application gives different security challenges to business and security professionals in that they expose the integrity of their data to the public. The increased accessibility to the corporation's system through websites has an impact on ever increasing need for the security of computer.

Web based attacks are the topmost among all the risks associated with integrity, confidentiality and availability. Web based attacks such as SQL injection i.e. SQLI, Cross-site scripting i.e. XSS etc. focuses on a web based application layer 7 of the OSI reference model. Application vulnerabilities could give the way to malicious end clients to break a framework's protection mechanism normally to exploit or access private data or framework resources.

The number of assaults are increasing day by day. Many endeavors have been made to discover solution for the issue. The best arrangement is to create the program in a safe way. Many archives have been distributed in regard to secure advancement of web based applications although very little has managed. Web engineers are not yet security mindful, and the issues keep on appearing. Accordingly, security administrators are continuously searching for different measures that can be taken against this issue. Developers are not yet security aware, and the issues continue to appear. Thus security experts are constantly looking for some other countermeasures which can be considered against the problem.

The possibility of failure of security in web application is high in today's web world. This thesis largely focuses on the challenges found in the security of web attacks. To overcome these challenges, the work presented in this thesis concentrates on designing and developing a security system. The proposed security system is a hybrid system which is a combination of web based attacks. This hybrid security system prevents the most commonly found serious and dangerous web attacks which are Cross Site Script i.e. XSS, SQL Injection i.e. SQLI and Cross-Site Request Forgery i.e. CSRF. The security system is developed in PHP. This proposed hybrid security system prevents the most commonly found serious and dangerous web attacks mentioned above in a more efficient way by reducing the drawbacks of the existing techniques given by many researchers which are being observed and thereby to improve performance. To analyze the efficacy of the methodology which has been proposed, the results are calculated on different set of PHP applications. All proposed techniques in this thesis have been validated and the results which are obtained shows the efficiency of the proposed security system.

TABLE OF CONTENTS

Dedication	ii
Candidate's Declaration	iii
Certificate	iv
Acknowledgement	v
Abstract	vi
Table of Contents	viii
List of Tables	xii
List of Figures	xiii
List of Abbreviations	xvi
CHAPTER I: INTRODUCTION	1-8
1.1 Introduction to Information Security	1
1.2 Web Applications and its Security	2
1.3 Motivation and Research Objective	4
1.4 Challenges related to Security of Web Attacks	6
1.5 Organization of Thesis	8
CHAPTER II: WEB ATTACKS AND VULNERABILITIES	9-22
2.1 Introduction	9
2.2 Vulnerability	9
2.3 Top Ten Vulnerabilities	9
2.3.1 Cross Site Script	12
2.3.1.1 Non Persistent XSS	13
2.3.1.2 Persistent XSS	13
2.3.2 SQL Injection	16
2.3.2.1 Tautology based SQL Injection	17
2.3.2.2 Union Query based SQL Injection	18
2.3.2.3 Stored Procedure based SQL Injection	18
2.3.2.4 Blind Injection based SQL Injection	19
2.3.2.5 Piggy Backed Query based SQL Injection	19
2.3.3 Cross Site Request Forgery	20

2.4 Conclusion	22
CHAPTER III: LITERATURE SURVEY	23-52
3.1 Open Web Application Security Project	23
3.2 Web Application Security Consortium	23
3.3 XSS Attack	24
3.3.1 Related Work for XSS Attack	25
3.4 SQL Injection Attack	36
3.4.1 Related Work for SQL Injection Attack	37
3.5 CSRF Attack	46
3.5.1 Related Work for CSRF Attack	46
3.6 Conclusion	51
CHAPTER IV: A HYBRID SECURITY SYSTEM FOR PREVENTION OF XSS,SQL,CSRF WEB ATTACK: PROPOSED APPROACH	53-78
4.1 Introduction	53
4.2 Abstract View of Proposed Hybrid Security System	54
4.3 Overall Architecture of Proposed Hybrid Security System	54
4.4 Phases of Hybrid Security System	56
4.5 Prevention of XSS Attack using Hybrid Security System	58
4.5.1 Scanning and Hotspot Identification Phase	58
4.5.2 Instrumentation Phase	58
4.5.3 Tag Attribute Model Phase	59
4.5.4 Validation and Error Report Phase	60
4.6 Prevention of SQL Injection Attack using Hybrid Security System	62
4.6.1 Scanning and Hotspot Identification Phase	62
4.6.2 Instrumentation Phase	62
4.6.3 SQL Query Model Phase	62
4.6.4 Validation and Error Report Phase	65
4.7 Prevention of CSRF Attack using Hybrid Security System	67
4.7.1 Scanning and Hotspot Identification Phase	67
4.7.2 Instrumentation Phase	67

4.7.3 Token Session Model Phase	67
4.7.4 Validation and Error Report Phase	68
4.8 Algorithms	69
4.8.1 Scanning and Hotspot Identification Algorithm	70
4.8.2 Instrumentation Algorithm	71
4.8.3 Model Generation Algorithm	72
4.8.4 XSS Attack Prevention Algorithm	74
4.8.5 SQL Injection Attack Prevention Algorithm	75
4.8.6 CSRF Attack Prevention Algorithm	76
4.8.7 Validation and Error Reporting Algorithm	76
4.9 Conclusion	77
CHAPTER V: IMPLEMENTATION AND EXPERIMENTAL ANALYSIS	79-124
5.1 Introduction	79
5.2 Implementation of Proposed Hybrid Security System	79
5.2.1 SQL Injection Vulnerability	79
5.2.1.1 Exploiting SQL Injection Vulnerability	80
5.2.1.2 Preventing SQL Injection Vulnerability	83
5.2.2 XSS Vulnerability	96
5.2.2.1 Exploiting XSS Vulnerability	96
5.2.2.2 Preventing XSS Vulnerability	100
5.2.3 CSRF Vulnerability	113
5.2.3.1 Exploiting CSRF Vulnerability	113
5.2.3.2 Preventing CSRF Vulnerability	115
5.3 Experimental and Comparative Analysis	120
5.3.1 Test Input Generation	121
5.3.2 Web Application Testing and Results	121
5.3.3 Comparative Analysis	122
5.4 Conclusion	124
CHAPTER VI: CONCLUSIONS AND FUTURE SCOPE	125-128
6.1 Conclusion	125

6.2 Benefits of Proposed Design	126
6.3 Future Scope	127
REFERENCES	129
PROFILE OF RESEARCH SCHOLAR	141
LIST OF PUBLICATIONS OUT OF THESIS	142

LIST OF TABLES

Table 2.1	Top10 web application vulnerabilities by OWASP	9
Table 2.2	Script tag based XSS attack	13
Table 2.3	Image tag based XSS attack	14
Table 2.4	Iframe tag based XSS attack	14
Table 2.5	Object tag based XSS attack	15
Table 2.6	Frame tag based XSS attack	15
Table 2.7	Div tag based XSS attack	15
Table 2.8	Tautology based SQLI attack	17
Table 2.9	Union Query based SQLI attack	18
Table 2.10	Stored Procedure based SQLI attack	19
Table 3.1	Literature survey of XSS attack	30
Table 3.2	Literature survey of SQL Injection attack	41
Table 3.3	Literature survey of CSRF attack	48
Table 4.1	Tag-Attribute model for static mode	59
Table 5.1	Experimental analysis for SQLIA	121
Table 5.2	Experimental analysis for Cross site script attack	122
Table 5.3	Experimental results for Cross site request forgery attack	122
Table 5.4	Comparative analysis of different techniques/approaches	123

LIST OF FIGURES

Figure 1.1	Web Application Architecture	3
Figure 2.1	View of XSS	12
Figure 2.2	Data flow using malicious SQL query	16
Figure 2.3	Series of action between Browser and Trusted-site	21
Figure 2.4	Series of action during CSRF attack	22
Figure 4.1	Architecture of Proposed Hybrid Security System	55
Figure 4.2	Prevention of XSS attack	61
Figure 4.3	SQL- query model during static mode	63
Figure 4.4	Tautology based SQL-query model during dynamic mode	62
Figure 4.5	Union query based SQL-query model during dynamic mode	64
Figure 4.6	Stored Procedure based SQL-query model during dynamic mode	64
Figure 4.7	Blind Injection based SQL-query model during dynamic mode	65
Figure 4.8	Piggy-backed query based SQL-query model during dynamic mode	65
Figure 4.9	Prevention of SQL Injection attack	66
Figure 4.10	Prevention of CSRF attack	69
Figure 4.11	Algorithm of Scanning and Hotspot identification for SQLI, CSRF and XSS	70
Figure 4.12	Algorithm of Instrumentation for SQL Injection, XSS and CSRF attack	71
Figure 4.13	Algorithm of Model Generation for SQL Injection, XSS and CSRF attack	72
Figure 4.14	Algorithm for Prevention of XSS attack	74
Figure 4.15	Algorithm for Prevention of SQL Injection attack	75
Figure 4.16	Algorithm for Prevention of CSRF attack	76
Figure 4.17	Algorithm for Validation and Error Report	76
Figure 5.1	Snapshot 1 of User Login Input Page	80
Figure 5.2	Snapshot 2 of User Login Page with Legitimate Input	81

Figure 5.3	Snapshot 3 of Output showing successful login	82
Figure 5.4	Snapshot 4 of User Login Page with Special Character Input	82
Figure 5.5	Snapshot 5 of Output showing successful login with special character	83
Figure 5.6	Snapshot 6 of SQL form to enter web application path	84
Figure 5.7	Snapshot 7 of Output generated after completion of step2	84
Figure 5.8	Snapshot 8 of Output generated after completion of step3	85
Figure 5.9	Snapshot 9 of Output generated after completion of step4	85
Figure 5.10	Snapshot 10 of Output generated after completion of step5	86
Figure 5.11	Snapshot 11 of User Login Page with legitimate input	87
Figure 5.12	Snapshot 12 of Output showing successful login	88
Figure 5.13	Snapshot 13 of User Login Page with Tautology based non-legitimate input	89
Figure 5.14	Snapshot 14 of Output showing sql injection attempted	89
Figure 5.15	Snapshot 15 of User Login Page with union query based non-legitimate input	90
Figure 5.16	Snapshot 16 of Output showing sql injection attempted	91
Figure 5.17	Snapshot 17 of User Login Page with blind injection based non-legitimate input	92
Figure 5.18	Snapshot 18 of Output showing sql injection attempted	92
Figure 5.19	Snapshot 19 of User Login Page with stored procedure based non- legitimate input	93
Figure 5.20	Snapshot 20 of Output showing sql injection attempted	94
Figure 5.21	Snapshot 21 of User Login Page with Piggy-backed query based non- legitimate input	95
Figure 5.22	Snapshot 22 of Output showing sql injection attempted	96
Figure 5.23	Snapshot 23 of User Login Page with legitimate input	97
Figure 5.24	Snapshot 24 of Output showing successful login	98
Figure 5.25	Snapshot 25 of User Login Page with malicious input	99
Figure 5.26	Snapshot 26 of Output showing vulnerability to XSS attack	100

Figure 5.27	Snapshot 27 of XSS form to enter web application path	101
Figure 5.28	Snapshot 28 of Output generated after completion of step2	101
Figure 5.29	Snapshot 29 of Output generated after completion of step3	102
Figure 5.30	Snapshot 30 of Output generated after completion of step4	102
Figure 5.31	Snapshot 31 of User Login Page with legitimate input	103
Figure 5.32	Snapshot 32 of Output showing successful login	104
Figure 5.33	Snapshot 33 of User Login Page with malicious script tag input	105
Figure 5.34	Snapshot 34 of Output showing xss attack attempted	105
Figure 5.35	Snapshot 35 of User Login Page with malicious source tag input	106
Figure 5.36	Snapshot 36 of Output showing xss attack attempted	107
Figure 5.37	Snapshot 37 of User Login Page with malicious body tag input	107
Figure 5.38	Snapshot 38 of Output showing xss attack attempted	108
Figure 5.39	Snapshot 39 of User Login Page with malicious image tag input	109
Figure 5.40	Snapshot 40 of Output showing xss attack attempted	109
Figure 5.41	Snapshot 41 of User Login Page with malicious iframe tag input	110
Figure 5.42	Snapshot 42 of Output showing xss attack attempted	111
Figure 5.43	Snapshot 43 of User Login Page with malicious div tag input	111
Figure 5.44	Snapshot 44 of Output showing xss attack attempted	112
Figure 5.45	Snapshot 45 of User Login Page with malicious embed tag input	112
Figure 5.46	Snapshot 46 of Output showing xss attack attempted	113
Figure 5.47	Snapshot 47 of User Login Page with legitimate input	114
Figure 5.48	Snapshot 48 of Output showing successful login	115
Figure 5.49	Snapshot 49 of CSRF form to enter web application path	115
Figure 5.50	Snapshot 50 of Output generated after completion of step2	116
Figure 5.51	Snapshot 51 of Output generated after completion of step3	117
Figure 5.52	Snapshot 52 of Output generated after completion of step4	117
Figure 5.53	Snapshot 53 of User Login Page with legitimate input	118
Figure 5.54	Snapshot 54 of Output showing successful login	119
Figure 5.55	Snapshot 55 of User Login Page with malicious input	119
Figure 5.56	Snapshot 56 of Output showing csrf attack attempted	120

LIST OF ABBREVIATIONS

WWW	World Wide Web
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
CGI	Common Gateway Interface
ASP	Active Server Pages
PHP	Hypertext Preprocessor(earlier called, Personal Home Page)
OWASP	Open Web Application Security Project
DOS	Denial Of Service
OSI	Open Systems Interconnection
SQL	Structured Query Language
SQLI	Structured Query Language Injection
SQLIA	Structured Query Language Injection Attack
XSS	Cross Site Script
CSRF	Cross Site Request Forgery
UI	User Interface
URL	Uniform Resource Locator
HTTP	Hyper Text Transfer Protocol
WASC	Web Application Security Consortium
CFG	Control Flow Graph
W3C	World Wide Web Consortium
FSA	Finite State Automata
XML	Extensible Markup Language
XSD	XML Schema Definition
API	Application Program Interface
JSP	Java Server Pages
LDAP	Lightweight Directory Access Protocol
IDS	Intrusion Detection System
JDBC	Java Database Connectivity
IP	Internet Protocol

CHAPTER I

INTRODUCTION

1.1 INTRODUCTION TO INFORMATION SECURITY

Information is an important resource and achieving a proper security level can be viewed as basic keeping in mind the end goal to keep up a focused edge. The advancement of the web has established a framework for the improvement and use of new classifications of systems of information technology working on the web[1]. The use of internet has been tremendously grown for accomplishing important task, such as online shopping, e-banking, e-reservation, e-governance etc.

With the advancement of WWW the organizations are starting to get more refined about how they utilize their website. Nowadays, the web has turned out to be extremely fundamental need of the general public, offices and objectives of web clients. WWW has developed quickly into a flexible stage for a wide range of computation, dynamically picking up support for information passage, client side scripting, and application-particular system dialogues.

Vulnerabilities has surfaced the different systems, with the expansion in web-human interaction. With the quickly developing technology, the simplicity of accessing through web applications has changed the conventional perspective of an organization completely. Hacking of webpages of web application keep on gaining notoriety as hackers are exposing vulnerabilities over all geographies and crosswise over different sorts of web technologies. Hackers are always exploring different avenues regarding an extensive variety of assaulting strategies to trade off sites and hack delicate information, for example, MasterCard number, social security number and other individual data. In this way, there is a prime need to secure the website against hackers and attackers.

Security has turned into a noteworthy aspect to the Internet world as internet age is rising day by day. Security of the web application is important of the fact that now every one of the activities like sharing, communication, sharing the assets, e-administering, online managing a back

account, online business, social communication, payment of different utilities bills etc. done on the internet.

With this rapidly changing technology, there are new risks potentially more damaging risks that will undoubtedly occur, and organizations have to continue to fight with those risks. As our daily lives are more becoming depending on the internet, it is necessary to understand broader views of the cyber threat. This rise of the cyber threat issues are hurting new economy. Despite all newly adopted web technology for e-services, cyber criminals are constantly developing and trying out various inbuilt advanced hacking tools to perform more dangerous and undetectable attacks.[1]

1.2 WEB APPLICATIONS AND ITS SECURITY

A web based application is accessible through the internet. In the past, it has been observed a number of creative and lethal attacks. Web attacking is gaining popularity because attackers are exploiting vulnerabilities throughout. According to a survey, the total numbers of vulnerabilities are throughout stabilized but web application[2] related vulnerabilities are continued to hover around 75% of total vulnerabilities. Web application carries sensitive data and they are accessible 24x7. This may give greater opportunity to the hacker to execute malicious activity. Although financial gain is the primary motivating factor for the web hacking, it is also experienced attacks to steal intellectual property, student's record and defacing the websites. Due to tremendous increase in the web application vulnerabilities and their vast impact on the business, many security organizations evaluate the security of their existing web applications.

Web based applications are executed on a web browser. They work on Three-tier architecture[3][4] which is mentioned below in Figure1.1.

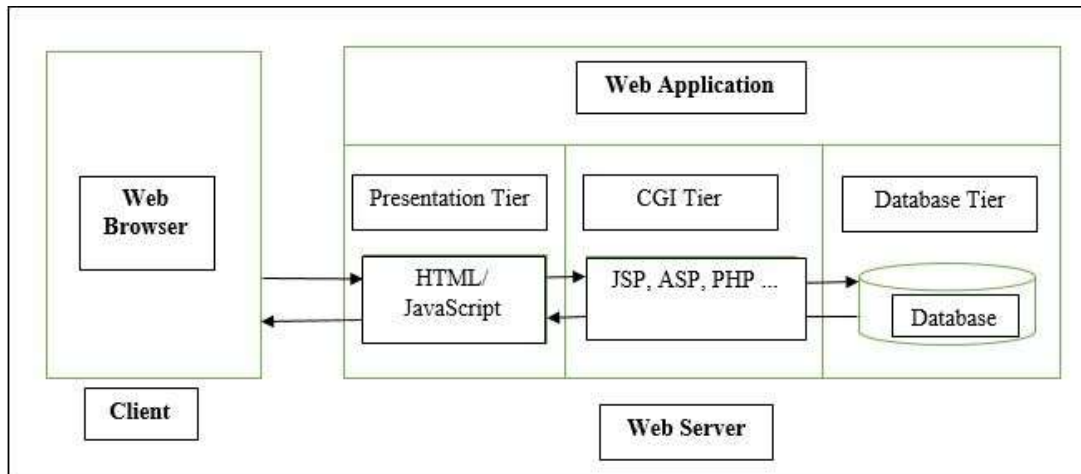


Figure1.1 Web application architecture

1) Presentation Tier:

It gets client's information and demonstrate output of prepared information to client. It is considered as GUI. The presentation tier is associated with user by Flash, HTML, JavaScript etc.

2) CGI Tier:

It is called Server Script which is placed between database tier and presentation tier. Data which is given by client is handled and stored inside the database. Data which is stored is then returned back to this tier which is then further returned back to presentation layer as output. Data preparing inside the application is processed at CGI layer. The data is modified using different scripting languages for example ASP.Net , PHP etc.

3) Database Tier:

This tier stores the entire data and deals with majority of prepared user data. Every sensitive data of web based application is put away and saved inside the database. This tier is incharge of granting access to legitimate clients and denial of fake clients from database.

The input validation problem is created when the data flows through tiers for the server. The input should be checked or modified before processing. The web application's security is compromised when there is a failure while checking the input.

Web application gives different security challenges to business and security professionals in that they expose the integrity of their data to the public. According to OWASP[5], increased accessibility to the corporation's system through websites has also had an impact on ever increasing need for the security of computer.

1.3 MOTIVATION AND RESEARCH OBJECTIVE

Implementation of software bugs is behind most security vulnerabilities detailed today. 20% of the vulnerabilities are named DOS assaults, 30% are because of outline mistakes, and nearly everything else is because of usage blunders and these indications are done by the study of vulnerabilities. Among errors from implementation, 84% are because of summed up injection vulnerabilities that permit an attacker to alter the estimations of security-sensitive factors utilizing crafted inputs to programs that are vulnerable. Web application attacks are the topmost among all the risks associated with integrity, confidentiality and availability[6]. The reason for a web application attack is altogether unique in relation to different attacks. They concentrate on application and capacities on layer 7 of OSI reference model. The vulnerabilities provide the way to malicious users to break a framework's security mechanism commonly to exploit or access private data or framework resources.

Hacking permits hacker to pick up access over the database and subsequently, a hacker might have the capacity to change information. The vast majority of the day by day activities rely on database driven web applications as a result of expanding task, such as banking etc. For performing different tasks, for example, paying of bills etc. information should be confidential. Web applications are often vulnerable to perform attacks, which further give hackers to easy access to the database.

In light of the expanded number of assaults exploiting, many endeavors have been made to discover solution for the issue. The best arrangement is to create the programs in a safe way. Many archives have been distributed in regards to secure advancement of web based applications with taking center on database, although very little has managed. Web engineers are not yet security mindful, and the issues keep on appearing. Accordingly, security administrator continues searching for different measures that can be taken against this issue. Developers are

not yet security aware, and the issues continue to appear. Thus security experts constantly looking for some other countermeasures which can be considered against the problem.

Although there exist many prevention techniques in the literature, there are certain points where the existing methods can be optimized or there is requirement of new technique. A critical study of literature available in the area of web attacks has been performed and some shortcomings were identified which motivated to pursue this research work.

- Incomplete implementations
- False alarms
- Run time overhead
- Complex framework

The objective of the research work is to design a security system for web attacks. This security system is a hybrid system which prevents the most commonly found serious and dangerous web attacks which are Cross Site Script i.e. XSS, SQL Injection i.e. SQLI, Cross-Site Request Forgery i.e. CSRF. The security system is developed in PHP. This hybrid security system uses combined analysis i.e. combination of static analysis and dynamic analysis both. This hybrid security system prevents the most commonly found serious and dangerous web attacks mentioned above in a more efficient way by reducing the drawbacks of the existing techniques given by many researchers which are being observed and thereby to improve performance. To achieve this objective, the work on the following goals has been performed in this thesis:

- To design a Security system for the most commonly found serious and dangerous web attacks on web applications.
- To design a framework for Security system
- To propose prevention techniques for the most commonly found serious and dangerous web attacks such are Cross Site Script i.e. XSS, SQL Injection i.e. SQLI, Cross-Site Request Forgery i.e. CSRF on web applications.
- To perform validation and to generate results and error report.
- To perform evaluation of Security system with a set of web applications of different complexities.

- To perform experimental and comparative analysis of Security system.

1.4 CHALLENGES RELATED TO SECURITY OF WEB ATTACKS

Based on the motivation and objectives defined in the previous section, this section discusses the challenges and their solutions while considering security of web attacks.

Static and Dynamic code analyzing: Static analysis[7,8] involves no dynamic execution of the software under test and can distinguish conceivable deformities in an early stage, before running the program. Static analysis is done subsequent to coding. However, there are certain drawbacks of static code analysis-

- If it is done manually it is time consuming.
- False positives and false negatives are created by automated tools.
- There are professionally untrained users for doing static code analysis.
- In the runtime condition, it does not find any vulnerabilities.

In contrast to Static code analyzing, where code is not executed, Dynamic code analyzing[7,9] is based on the system execution, often using tools. Dynamic code analyzing recognizes vulnerabilities in a runtime environment. However, there are certain drawbacks of dynamic code analysis-

- A false sense of security provided by automated tools that everything is being addressed.
- Cannot guarantee the full test extent of the source code.
- Automated instruments make false positives and false negatives are created by automated tools.
- It is harder to follow the weakness back to the right area in the code, taking more time to settle the problem.

Solution: *A hybrid security system has been proposed which uses combined analysis i.e static code analysis and dynamic code analysis both so as to overcome the limitations of dynamic code analysis as mentioned above.*

Complex Framework: Complex framework for web application describes the complexity of the framework in terms of space and time requirements. While designing a software or while writing a piece of code, its efficiency is determined by the complexity of web framework. Complex framework gives a unit to measure the efficiency of the framework. This issue is being observed in the existing techniques given by the researchers during the study of literature available.

***Solution:** This issue has been resolved by proposing a modular approach which uses different modules for performing different tasks in a simplified manner. The framework proposed is simple, efficient, easy to understand and implemented in an efficient manner. The framework proposed is designed in such a way that different types of web based attacks can be combated more efficiently and the identity and credentials of the user are prevented from being exposed to the unauthorized intruder.*

Incomplete Implementation: It has been observed during the study of literature available that most of the existing techniques proposed by the researchers are not covering all the types of a specific web attack i.e partial implementation is done in most of the cases.

***Solution:** This issue has been resolved by considering all the types of a specific attack in order to ensure complete implementation.*

Run time overhead: Run time overhead refers to the processing time delay due to the malicious activity. It occurs when that there is an unusual delay. [10] Processing time is the time calculated from the start of the application to the minute it ends. It is an important challenge related to security of web attacks.

***Solution:** This issue has been reduced by proposing a hybrid system which uses modular approach for performing different tasks in a simplified manner. The framework proposed is simple and implemented in an efficient manner so as to reduce runtime overhead.*

False alarm: False alarms happen when a query has been erroneously delegated malevolent. [11]False alarm is classified into false positive and false negative. False positive means detection of attack even if it does not exist. False negative means non-detection of attack even if it exists.

It has been observed during the study of literature available that most of the existing techniques proposed by the researchers are showing false alarms.

***Solution:** This issue has been resolved by proposing a hybrid system which is designed very efficiently so as to eliminate false alarms by checking with maximum possible data set.*

1.5 ORGANIZATION OF THESIS

This thesis has been organized in the following chapters:

Chapter 2 discusses about the web attacks and vulnerabilities. It begins by presenting an introduction of web attacks. This chapter later on discusses the various types of vulnerabilities present in the web applications.

Chapter 3 provides an insight into the literature review which motivated this research work. The detailed literature survey is discussed here. This chapter provides the backdrops of existing work and further explores the possibility of improvement.

Chapter 4 furnishes a security system which is presented in the light of drawbacks of the existing work. The proposed security system is a hybrid system which works in four phases. It begins by discussing the overall architecture of security system. It further discusses the prevention of most commonly found serious and dangerous web attacks which are Cross Site Script i.e. XSS, SQL Injection i.e. SQLI, Cross-Site Request Forgery i.e. CSRF using hybrid system.

Chapter 5 provides the implementation of security system. Later on experimental analysis and comparative analysis for different web attacks is performed. To analyze the efficiency of proposed method, results are evaluated on different web based applications.

Chapter 6 concludes outcome of the work proposed in this thesis. It also endeavors to explore the possibilities of future research work based on the proposed design.

CHAPTER II

WEB ATTACKS AND VULNERABILITIES

2.1 INTRODUCTION

Web based attacks are the topmost among all the risks associated with integrity, confidentiality and availability [6]. Web based attacks such as SQL injection i.e. SQLI, Cross-site scripting i.e. XSS etc. focuses on a web based application layer 7 of the OSI reference model. Application vulnerabilities could give the way to malicious end clients to break a framework's protection mechanism normally to exploit or access private data or framework resources.

2.2 VULNERABILITY

A web based vulnerability is a shortcoming within the application, which could be an outline defect or a implementation bug which permits a hacker to harm the stakeholders of web based application. Stakeholders incorporate the application proprietor, application clients, and different entities that depend on the application.

2.3 TOP TEN VULNERABILITIES

OWASP[5] classifies top level web application based vulnerabilities. The top ten web application based vulnerabilities listed by OWASP is shown below in Table2.1.

Table2.1. Top 10 Web Application Vulnerabilities by OWASP

Vulnerability Rank	Vulnerability Name	Description
1	SQL Injection Attack (SQLIA)	SQLIA[5] happens when entrusted information is sent as a part of query or an sql command. The hacker's unfriendly information forces the

		interpreter to run malicious commands or getting unapproved information.[78][79]
2	Cross Site Scripting(XSS)	XSS[5] defects happen when a web based application takes entrusted information and forwards it to a web browser without proper validation. This attack permits hacker to run scripts in target's program which could capture user's session, destroy websites, divert the legitimate user to malicious websites etc.[77]
3	Cross Site Request Forgery(CSRF)	A CSRF[5] web vulnerability forces a signed on target's program which sends HTTP request which is forged. It includes victim's session cookie and consequently verification of information to a vulnerable webpage. This permits the hacker to compel the target.[75,80]
4	Broken Authentication and Session Management	Application function of web application are identified with validation [5,12] are regularly executed incorrectly, permitting hackers to compromise login credentials and other user's identities.
5	Insecure Direct Object References	It occurs when a programmer opens a link to an inward execution question, for example, a record, index etc. Due to the lack of protection, hackers can control these links to get to malicious data.[13]
6	Security Misconfiguration	A secure and protected configuration [5] is characterized for web based applications. The Misconfiguration can be avoided by presenting a repeatable procedure for software updates, patches, and hardened environment rules.[14]
7	Sensitive Data	Applications that don't utilize the cryptographic

	Exposure	protection conspire for sensitive data [5], for example, healthcare information, credit card, personal data, and verification details fall under this class. By actualizing the strong standard encryption or hashing calculation one can guarantee the security of information.[15]
8	Missing Function Level Access Control	Most of the web based applications confirm function level access permissions [5] before showing their usefulness visible in UI. An access control check is required when each module is accessed. If user requests are not checked, the hackers will have the capacity to access.[16]
9	Using components with known vulnerabilities	Modules like libraries , source code etc are often executed with full rights[5].If a vulnerable part is misused, such a hack can cause serious information loss or server takeover.[17]
10	Unvalidated Redirects and Forwards	Web based applications are generally redirect users to different pages to access entrusted information. Due to inappropriate validation, hackers can divert target to malicious websites so as to access unauthorized pages.[18]

The most commonly found serious and dangerous web based vulnerabilities as listed in OWASP are SQL injection i.e. SQLI, Cross Site Script i.e. XSS and Cross Site Request Forgery i.e. CSRF. These are described in the subsequent sections-

2.3.1 Cross-Site Script(XSS)

It is a kind of injection in which hacker injects his own script code into a vulnerable website page. At the point when a victim visits this infected page in the web application just by browsing the web site, his browser downloads the hacker code and automatically executes it by accessing any file[19]. A hacker can send a malicious script to a non-suspecting client utilizing XSS. The end client browser does not have the possibility that the script ought not be trusted, and thus run the script. The malicious JavaScript seems to be a legitimate component of the web application as per victim's program. The hacker would be able to access data i.e. cookies, session id etc after the running of malicious script. [20,21]

A Typical View of XSS is shown below in Figure2.1.

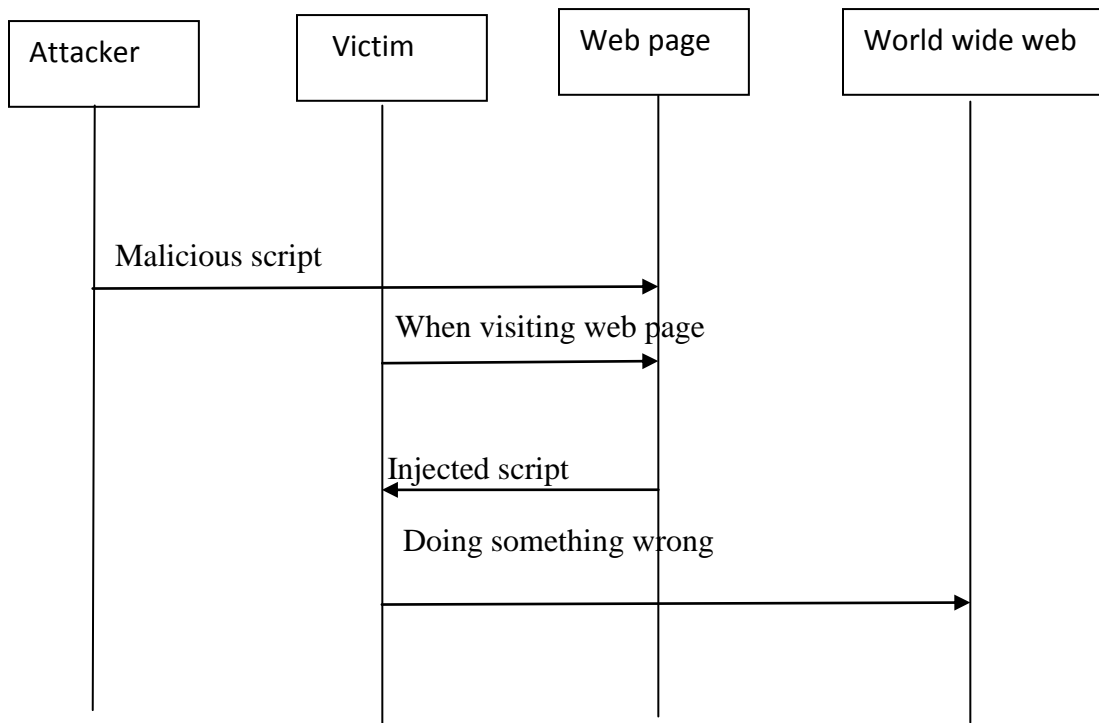


Figure2.1. View of XSS

Different types of XSS attacks are mentioned below.

2.3.1.1 Non-Persistent (or Reflected) XSS

This type of exploit targets web vulnerabilities which occurs when the data which is put together by the user is quickly handled by server to produce results . An exploit is successful if the script is forwarded to the web server which is further incorporated into the web page. Victims are targeted individually and no script is injected at the trusted site's server. This attack is delivered to victim through email or from some other website. The bait could be a URL pointing towards a trusted site, clicking which executes the malicious script. The injected attack is not stored within the web application itself and only users who opened a malicious link are victimized, hence, called non persistent XSS.

2.3.1.2 Persistent (or Stored) XSS

It occurs when the malicious script is submitted to a webpage of a website where it is stored for some time. The example of a hacker's most loved target includes web chatting sites etc. The unsuspecting client do not interact with extra website (e.g. a hacker website sent by email or any other social email clients), just simply to see the page containing the code. Malicious script is injected by the attacker at the trusted site's server. It could be present in the database, message forum or comment field. This type of attack does not require targeting victim individually and continue to attack victims when they request data associated with malicious script, hence, called persistent XSS.

The Tables mentioned below shows the different types of Tag based XSS attack possibilities.

Table2.2. Script tag based XSS attack

S.No.	Script tag based XSS attack
1.	< script > alert ('Cross Site Script i.e. Xss') </script>
2.	< script > alert("Cross Site Script i.e. Xss") </script>
3.	< script >alert("Cross Site Script i.e. Xss");</script>
4.	< script > alert (document.cookie()) ; </script>
5.	< script > alert(/Cross Site Script i.e. Xss") ; </script>
6.	< script > alert(/Cross Site Script i.e. Xss/) ; </script>

7.	< /script > <script> alert(/Cross Site Script i.e. Xss/); < /script >
8.	< script > < /script > <script> alert(Cross Site Script i.e. Xss); < /script >
9.	< script > "> < /script > alert(/Cross Site Script i.e. Xss/); < /script >

Table2.3. Image tag based XSS attack

S.No.	Image tag based XSS attack
1.	< img src = ' javascript: alert(' Cross Site Script i.e. Xss') ; ' > < /img >
2.	< img src = ' javascript:alert("Cross Site Script i.e. Xss") ; ' > < /img >
3.	< img src = ' javascript:alert("/ Cross Site Script i.e. Xss ") ; ' > > < /img >
4.	< img src = ' javascript:alert(/Cross Site Script i.e. Xss /) ; ' > < /img >
5.	< img src =" javascript:alert(' Cross Site Script i.e. Xss') ; " > < /img >

Table2.4. Iframe tag based XSS attack

S.No.	Iframe tag based XSS attack
1.	< iframe src = " javascript:alert('Cross Site Script i.e. Xss') " > < /iframe >
2.	< iframe src=" javascript:alert("Cross Site Script i.e. Xss") " > < /iframe >
3.	< iframe src=" javascript:alert("/Cross Site Script i.e. Xss") " > > < /iframe >
4.	< iframe src=" javascript:alert(/Cross Site Script i.e. Xss/) " > < /iframe >
5.	< iframe src=" javascript:alert(/Cross Site Script i.e. Xss/) "> < /iframe >

Table2.5. Object tag based XSS attack

S.No.	Object tag based XSS attack
1.	< object data="" javascript:alert('Cross Site Script i.e. Xss')" > < /object >
2.	< object data="" javascript:alert("Cross Site Script i.e. Xss")" > < /object >
3.	< object data="" javascript:alert("/Cross Site Script i.e. Xss")"> < /object >
4.	< object data="" javascript:alert(/Cross Site Script i.e. Xss/) " > < /object >
5.	< object data="" javascript:alert(/Cross Site Script i.e. Xss/) " > < /object >

Table2.6. Frame tag based XSS attack

S.No.	Frame tag based XSS attack
1.	<frame onclick=""javascript:alert('Xss')"></frame>
2.	<frame onmouseover=""javascript:alert('Xss')"></frame>
3.	<frame onmouseout=""javascript:alert('Xss')"></frame>
4.	<frame onclick=""javascript:alert("Xss")"></frame>
5.	<frame onmouseover=""javascript:alert("Xss")"></frame>
6.	<frame onmouseout=""javascript:alert("Xss")"></frame>
7.	<frame onclick=""javascript:alert(/Xss")"></frame>
8.	<frame onmouseover=""javascript:alert(/Xss")"></frame>
9.	<frame onmouseout=""javascript:alert(/Xss")"></frame>
10.	<frame onclick=""javascript:alert(/Xss/) "></frame>
11.	<frame onmouseover=""javascript:alert(/Xss/) "></frame>
12.	<frame onmouseout=""javascript:alert(/Xss/) "></frame>

Table2.7. Div tag based XSS attack

S.No.	Div tag based XSS attack
1.	< div style="" javascript:alert('Cross Site Script i.e. Xss')" > </div>
2.	< div style="" javascript:alert("Cross Site Script i.e. Xss")" > </div>
3.	< div style="" javascript:alert(/Cross Site Script i.e. Xss")" > < /div >
4.	< div style="" javascript:alert(/Cross Site Script i.e. Xss/) " > < /div >

5.	< div style=" background-image: url(javascript:alert(' Cross Site Script i.e. XSS'))" >
----	---

2.3.2 SQL Injection

SQLI is a standout amongst the most widely recognized major threat to database driven application security. [5,22] SQL injection is a method where malicious SQL queries are induced as an input so as to exploit the weakness present within database. The attacker tries to inject malicious data. The unauthorized access takes place after the execution of malicious input. It permits a hacker to pick up control over the database of an application and therefore, a hacker will be able to change the data. [23,24,25]

In database driven web applications, SQL queries join client provided information or content. If inclusion of client provided information is done in a risky way, then the web application ends up simply vulnerable to SQLIA. The SQLI vulnerabilities happen between CGI layer and Presentation layer. The information flow between every level utilizing malicious input information is shown in Figure2.2.

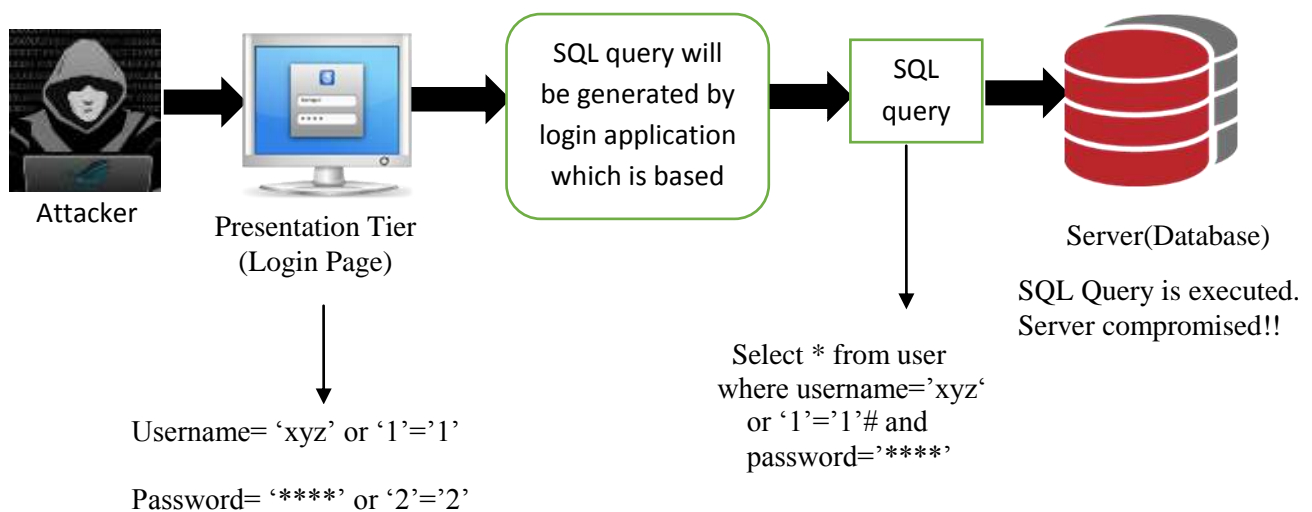


Figure2.2. Data flow using malicious SQL query

Different types of SQL Injection attacks are mentioned below.

2.3.2.1 Tautology based SQL injection

In this type of attack, the conditional statements are written in such a manner so that the query is always true. The attacker's aim is to extract the data from the database and to bypass the authentication.

For example-

```
SELECT * FROM EMPLOYEE WHERE USERNAME='alice' OR '1'='1'# AND
PASSWORD= '*****';
```

Following Table2.8 shows different Tautology based attack possibilities.

Table2.8.Tautology based SQLI attack

S.No.	Tautology based SQLI attack
1.	'name' OR '1'='1' #
2.	'name' OR '1'='1' --
3.	'name' OR '1'='1' ++
4.	'name' OR '1'!='0' --
5.	'name' OR '1'!='0' #
6.	'name' OR '1'!='0' ++
7.	' OR ' '=' --
8.	' OR ' '=' ++
9.	' OR ' '=' #
10.	'name' OR '1'='1' AND '0'='0' #
11.	'name' OR '1'='1' AND '0'='0' --
12.	'name' OR '1'='1' AND '0'='0' ++
13.	'name' OR '1' --
14.	'name' OR '1' #
15.	'name' OR '1' ++

2.3.2.2 Union query based SQL injection

In this type of SQLI, the operator ‘union’ is used to link malicious query with legitimate SQL query. The attacker’s aim is to extract the data from the database and to bypass the authentication .

For example-

```
SELECT * FROM EMPLOYEE WHERE USERNAME='alice' UNION SELECT * FROM  
EMPLOYEE AND PASSWORD= '*****';
```

Following Table2.9 shows different Union query based attack possibilities.

Table2.9.Union Query based SQLI attack

S.No.	Union query based SQLI attack
1.	'name' UNION select * from Users
2.	'name' UNION update Users set password='abc'
3.	'name' UNION drop table Users
4.	'name' UNION delete * from Users where username='alice'
5.	'name' UNION drop database system
6.	'name' UNION insert into Users values username='abc' and password = 'xyz'
7.	'name' UNION select count(*) from Users where username='abc'and password='xy'
8.	'name' UNION select count(*) from Users where username = 'abc' and password LIKE '%w%'
9.	'name' UNION select count(*) from Users

2.3.2.3 Stored Procedure based SQL injection

In this type of attack, the malicious actions are performed using built-in procedures. The attacker’s aim is to privilege escalation and to execute remote commands.

For example-

```
SELECT * FROM EMPLOYEE WHERE USERNAME=' name ' ; SHUTDOWN AND  
PASSWORD='*****';
```

Following Table2.10 shows different Stored Procedure based attack possibilities.

Table2.10. Stored Procedure based SQLI attack

S.No	Stored Procedure based SQLI attack
1.	'name' ; sp_monitor
2.	'name' ; sp_executesql drop table Users
3.	'name' ; sp_server_info
4.	'name' ; sp_tables
5.	'name' ; SHUTDOWN
6.	'name' ; sp_helptext
7.	'name' ; drop table Users

2.3.2.4 Blind Injection based SQL injection

In this type of attack, the hacker injects query to discover the vulnerabilities. Logical conclusions are made depending on true or false questions. DB schema is estimated by collecting responses of either true or false questions. The attacker's aim is to extract data, to discover schema and to identify injectable patterns .

For example-

```
SELECT * FROM EMPLOYEE WHERE USERNAME='name' AND 1=0 AND  
PASSWORD= '*****';
```

2.3.2.5 Piggy-backed query based SQL injection

In this type of SQLI , the hacker appends malicious query with legitimate query. At the time of execution of first query, second query simultaneously gets executed. The attacker's aim is to extract and modify data, Denial of service (DoS) .

For example-

```
SELECT * FROM EMPLOYEE WHERE USERNAME = 'alice' ; DROP TABLE USER AND  
PASSWORD='*****' ;
```

2.3.3 Cross-Site Request Forgery (CSRF)

This attack occurs when a non trusted website causes a client's web browser to permit a malicious activity on a trusted website. This is due to the forged HTTP request as it exploits the current client's session in the web browser. [5,26]A CSRF web attack requires inclusion of three things. A target client, a trustable web site, and a non trustable web site. The target client is currently holding an active session with a trustable site and in the meanwhile, the client visits a malicious or non trusted website. The non trustable or malicious web site injects a HTTP request for the trustable web site into the target client's session which compromises its integrity. These vulnerabilities permit a hacker to exchange money out from client's account, to collect client's email id, disregard client privacy etc. [27,28]

CSRF attack can be explained as follows-

Assume a client is signed in one of the trusted websites. In this manner, the web browser is utilized by the client session with the trustable website. The CSRF attack violates the connection between the client's browser and trustable website. At the point when client clicks this connection (while the authentication session is active), it traps the web browser to forward the request to client. As the web browser performs trusted activities, the request thus executed. It can be shown below using Figure2.3 and Figure2.4.

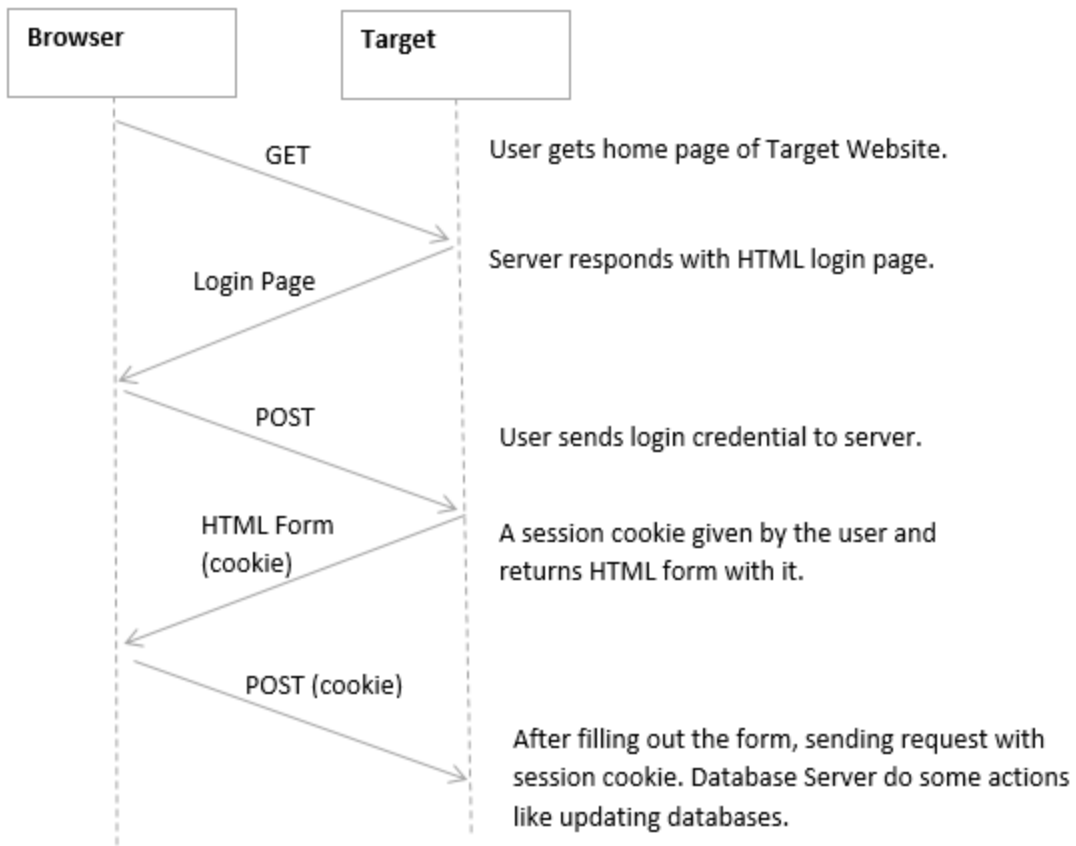


Figure2.3.Series of action between browser and trusted-site

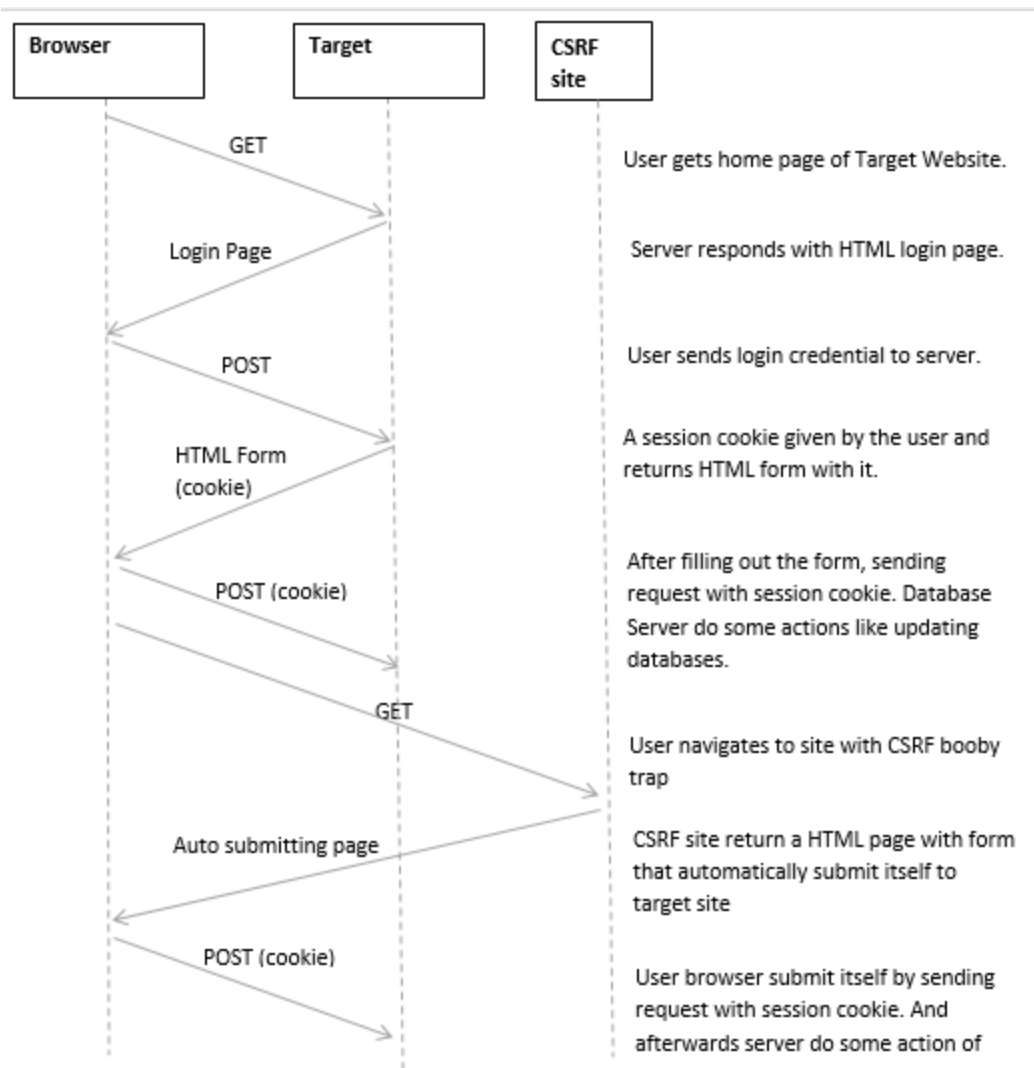


Figure2.4. Series of action during CSRF attack

2.4 CONCLUSION

There is a continuous growth of web attacks. Among the above mentioned top ten web based vulnerabilities as listed by OWASP[5]. SQL injection i.e. SQLI, Cross Site Script i.e. XSS, Cross Site Request Forgery i.e. CSRF are some of the most commonly found serious and dangerous exploits for web based applications. Hacking permits hacker to pick up access over the database and subsequently, a hacker might have the capacity to change information. Web applications are often vulnerable to perform attacks, which further give hackers to easy access to the database. In the light of the above, a detailed literature survey is discussed in the next chapter.

CHAPTER III

LITERATURE SURVEY

3.1 OPEN WEB APPLICATION SECURITY PROJECT (OWASP)

OWASP[5] looks to instruct designers, business owners, developers and architects about threats related to the most widely recognized web application based security flaws. OWASP supports open source as well as business related security items. It is an organization which lists topmost popular web application based security blemishes and gives suggestions for managing these vulnerabilities.

OWASP documents, tools etc. are classified into three main categories. Firstly, they are used to discover execution faults and security related policies. Secondly, they can be utilized to make preparations for execution faults and security related policies. Lastly, they can be used to include security related actions into application lifecycle administration i.e. ALM.

The OWASP lists current top ten web application based security flaws that are dangerous, alongside effective strategies for managing those flaws. OWASP is an organization that gives unbiased and reasonable, practical data about computer system and Internet applications. Members of the project incorporate different types of security experts from around the world who share their knowledge into vulnerabilities, attacks, threats and countermeasures.

3.2 WEB APPLICATION SECURITY CONSORTIUM(WASC)

Web Application Security Consortium i.e. WASC[29] is a collection of experts and specialists from industry that produces open source finest practices security guidelines for www. WASC is 501c3 non benefit comprising of collection of experts and specialists from industry that produces open source finest practices security standards for www. It was established in 2004 by Jeremiah Grossman (CTO, WhiteHat Security, Inc.) and Robert Auger (CGI Security).

As a dynamic community, WASC supports and encourages the exchange of thoughts and composes distinctive types of modern day projects. WASC reliably and occasionally releases security rules and many helpful documentation. Educational organizations, government organisations, application designers, security experts etc. use the items released by WASC to help and beat challenges displayed by web application related security.

3.3 XSS ATTACK

It is a kind of injection in which hacker injects his own script code into a vulnerable website page. At the point when a victim visits this infected page in the web application just by browsing the web site, his browser downloads the hacker code and automatically executes it with accessing any file[19]. A hacker can send a malicious script to a non-suspecting client utilizing XSS. The end client browser does not have the possibility that the script ought not be trusted, and thus run the script. The malicious JavaScript appears as a legitimate component of web application by the victim's program. Hacker would be able to access data i.e. cookies, session id etc after the running of malicious script. [20,21]

XSS attacks are mainly of two kinds as explained below.

- Stored/ Persistent XSS

It arises whenever the malevolent script is inputted to webpage of a website where it is stored for some time. The examples of a hacker's most loved target include web chatting sites etc. The unsuspecting client do not interact with extra website (e.g. a hacker website sent by email or any other social emails clients), just simply to see the page containing the code. Malicious script is injected by the attacker at the trusted site's server. It could be present in the database, message forum or comment field. This type of attack does not require targeting victims individually and continue to attack victims when they request data associated with malicious script, hence, called persistent XSS.

- Reflected/ Non-persistent XSS

This type of exploit targets web vulnerabilities which occur when the data which is put together by the user is quickly handled by server to produce result. An exploit is successful if the script is forwarded to the web server which is further incorporated into the web page. Victims are targeted individually and no script is injected at the trusted site's server. This attack is delivered to victims through email or from some other website. The bait could be a URL pointing towards a trusted site, clicking which executes the malicious script. The injected attack is not stored within the web application itself and only users who opened a malicious link are victimized, hence, called non persistent XSS.

3.3.1 Related Work for XSS Attack

- **Gupta et. al.[30]** presented a complete exploration of Cross site script attack. Its detection as well as prevention. Analysis of modern web application reveals some serious vulnerabilities. The proper handling of the input given by the user is of utmost importance, therefore vulnerabilities associated with these modern web application needs to be addressed. It concludes that there is a prime requirement to construct a safe framework by taking into account the existing methodologies.
- **Gupta et. al.[31]** explored XSS attack. XSS vulnerabilities were tested on Tomcat Apache Server and Web Goat with malicious script and cross site script attack mitigation was verified using encrypted forms of the script which is injected. It can be prevented by applying secure validation of input and proper sanitization of input given by user. In the end, requirement of automated process which will differentiate a malicious with a legitimate javascript, an efficient web crawler is used for web scanning and methodologies for whitelisting and blacklisting of code were laid down for the desired framework for the prevention of attack.
- **Saleh et.al.[32]** proposed a method for detection of web based vulnerabilities. It uses an algorithm named Boyer- Moore for string matching. It is seen that present scanners in market are limited by high false negatives. These limitations was taken care off in this method having less runtime overhead and great accuracy.

Though this method is not capable of detecting web application specific vulnerabilities but it can easily detect such vulnerabilities at web page level. It uses URL to detect such vulnerabilities. However, this method is not able to detect SQLI attacks and hence development of hybrid string matching algorithm is suggested for future work.

- **Gupta et. al.[33]** proposed a server side framework called XSS-SAFE. It is used to detect and to prevent cross-site scripting attack. It is due to the injection of JavaScript sanitization routines in the source code. This prototype framework was implemented in Java and evaluated on JSP programs. All known and unknown cross-site script attacks were successfully detected and mitigated. False negative rates were at 0% but false positive rate is 10-15%. This is because of the difference in the rules applied in case of different scripts. Hence false positive rates are needed to be minimized and framework modified to handle workflow violation attack as future work.
- **Salas et.al.[34]** used WSInject as a tool to analyze web services. It is a fault injection tool which unlike other security testing tools have different scenarios for multiple attacks. They compared their results with one of the vulnerability scanner named soapUI. They analyzed that soapUI is less efficient as compared to WSInject. XSS attack was performed by using WSInject tool. Afterwards they generated various attack scenarios. Finally they performed attacks with this tool. The analysis was verified by using WSInject. A number of vulnerabilities were reduced by the security tool proposed.
- **Fabien et. al.[35]** presented a methodology in which test inputs are created by combining model inference as well as evolutionary fuzzing methodology. By using this, the web injection vulnerabilities can be detected. Knowledge of application behaviour is obtained by model inference. GA i.e. Genetic Algorithm is used to generate input. The inputs are generated automatically having good fitness values towards causing an instance for the vulnerability which is given. An

automated XSS search technique was proposed which uses model inference as well as evolutionary fuzzing methodology for the creation of test cases.

- **Yu sun et.al.[36]** for defense against cross site scripting attacks proposed a model checking method. Bugs present in the e-commerce website were found and counter examples were shown by model checking. Proposed an automatic modeling algorithm for the HTML code and presented the case of performance of the algorithm.
- **Lwin Khin et.al.[37]** proposed different input sanitization strategies into various types and proposed a set of static code attributes. They have utilized data mining strategies to anticipate SQLI and XSS vulnerabilities. They have explained classification schemes depending on CFG (Control Flow Graph) for web based applications. They have implemented a tool called PhpMiner. It is utilized to extract the information and proposed characteristics from PHP programs.
- **Lwin Khin et.al.[38]** proposed various XSS exploits techniques that are similar to SQLI. The attacks are caused due to the improper sanitization of user input. They have also proposed various types of XSS defensive techniques such as detection of web vulnerabilities, prevention of attack at runtime etc.
- **Avancini et. al.[39]** proposed a search based methodology for security testing of web based applications. They exploited static based analysis to detect XSS vulnerabilities. They have used genetic algorithm based approach. Search based test cases are utilized by developers to resolve security related issues. They have implemented this methodology in a model and tested on PHP based applications.
- **Rattipong et. al.[40]** presented a methodology for protection of cookies from XSS attack. This method changes cookies in a manner so that the cookies become unworkable for the cross site script attacks. Dynamic Cookie Rewriting technique is executed for web applications. Prior to the sending of cookie to browser, the cookies are removed with randomized value. The randomized value is kept by the

browser by not keeping the unique value inputted by the browser. This procedure is tested on HTTP connections.

- **Adam et. al.[41]** created an automatic system which exposes SQLI and XSS vulnerabilities through generating input test cases. In this technique, test inputs are created and changed to deliver concrete exploits and track corruptions through execution. The technique addresses second order XSS attack, it makes genuine attack vectors, no modification of application code is required. The proposed methodology was executed for PHP applications by a tool named Ardilla, which can make input for single script at once but it is not able to simulate sessions(i.e. applications which involve multiple pages user-interaction). Ardilla cannot produce attacks for sink.
- **Mike et.al.[42]** proposed an XSS defensive methodology, in spite of behaviour anomalous browser it was intended to be effective in existing browser systems. The proposed approach minimizes trust placed on system browser for interpreting non trustable content. BLUEPRINT is a tool used to implement this approach and it was integrated with a few popular web applications. It was a strong way to prevent cross-site scripting attacks which was effective upon 96% of the system browser.
- **Wassermann et. al.[43]** presented a static analysis which specifically addresses weak or missing input validation for discovering cross-site scripting vulnerabilities. This approach combines data obtained through corrupted information flow with string analysis. Due to the numerous ways to invoke the Java interpreter input validation is troublesome. This approach confronts a similar obstacle that statistically check vulnerabilities. This is addressed by formalizing a policy which is given by W3C. An approach to find cross site script web based vulnerabilities because of non-checking of malicious information and insufficiently checked entrusted data is proposed in this paper. The approach is

divided into two parts. One is adjusted string analysis used for the tracking of entrusted substring values. Another is to check endowed scripts.

- **Shanmugam et. al.[44]** proposed an approach to detect behaviour based anomaly which presents a security layer on top of the websites, due to which whenever new threats appears the mechanisms are changed but the current websites stay unmodified. Also, to reduce processing time application parameters are acquainted, this approach gives security to websites by permitting tags to be entered in the websites. To decrease processing time, it uses whitelist security model due to which it is not prone to zero-day attacks.
- **Qianjie Zhang et. al.[45]** provided a prevention mechanism which prevents Cross-site Scripting (XSS) attacks using the execution flow mechanism for JavaScript running on client side. Firstly client side behavior of Ajax applications under normal circumstances is modeled as finite-state-automata(FSA) and deployed in proxy mode. Before running any JavaScript on client side it is analyzed against this model and only on conformation with this model it is allowed to run. It has been evaluated against various real life applications, and results show protection against several XSS attacks with acceptable performance overhead.
- **Shanmugam et. al.[46]** proposed a technique which presents a separate security layer above existing web applications which gives flexibility to these application not to be modified but yet to provide sufficient mechanism to prevent any new security threat. This layer uses signature based misuse detection. For analysis of this technique, vulnerable inputs from black hat hacker site were considered and run on JBoss server.
- **Shanmugam et. al.[47]** provided a solution for prevention against Cross Site Scripting (XSS) attack which is independent of any particular language implementation and is based on service oriented architecture on which various web applications architecture are based. XSS blocker is has independence from

any particular platform as well as language because development of interfaces such as converter as well as validator is required to be designed. The task is minimal. Another part of blocker like XML and XSD can also be easily consumed using APIs. A very less effort is required in executing this technique on web applications which are existed already. Evaluation of this approach with JSP/servlet application on JBoss web application server is very efficient and effective.

- **Shiuh-Jeng Wang et. al.[48]** provided a scheme that can be used to collect digital evidences on web systems after occurrence of XSS attack which then can be presented in court as evidence and help to recreate the crime-venue in crime case. Stolen personal information is also investigated in this scheme. Also a management strategy has also been provided to prevent XSS attack from network intrusion. Here the new methods of cross site script intrusion which uses HTTP as well as properties related to cross-platform is discussed .

The detailed literature survey of XSS attack can be shown below using Table3.1.

Table3.1. Literature Survey of XSS attack

S.No.	Author	Publisher/Year	Description
1.	Gupta et. al.[30]	Springer /2015	Authors presented a complete exploration of Cross site script attack, its detection as well as prevention. Analysis of modern web application reveals some serious vulnerabilities. The proper handling of the input given by the user is of utmost importance, therefore vulnerabilities associated with the modern web application needs to be addressed.
2.	Gupta et. al.[31]	Taylor & Francis /2015	Authors explored XSS attack. XSS vulnerabilities were tested on Tomcat Apache Server and Web Goat with malicious script and cross site script attack mitigation was verified using

			encrypted forms of the script which is injected. It can be prevented by applying secure validation of input and proper sanitization of input given by user.
3.	Saleh et.al.[32]	Elsevier /2015	Authors proposed a method for detection of web based vulnerabilities. It uses an algorithm named Boyer- Moore for string matching. It is seen that present scanners in market are limited by high false negatives. These limitations were taken care off in this method having less runtime overhead and great accuracy. It uses URL to detect such vulnerabilities. However, this method is not able to detect SQLI attacks and hence development of hybrid string matching algorithm is suggested for future work.
4.	Gupta et. al.[33]	Springer /2016	Authors proposed a server side framework called XSS-SAFE. It is used to detect and to prevent cross-site scripting attack. It is due to the injection of JavaScript sanitization routines in the source code. This prototype framework was implemented in Java and evaluated on JSP programs. False negative rates were at 0% but false positive rate is 10-15%. Hence false positive rates are needed to be minimized and framework modified to handle workflow violation attack as future work.
5.	Salas et.al.[34]	Elsevier /2014	Authors used WSInject as a tool to analyze web services. It is a fault injection tool which unlike other security testing tools have different scenarios for multiple attacks. They compared their

			<p>results with one of the vulnerability scanner named soapUI. They analyzed that soapUI is less efficient as compared to WSInject. Afterwards they generated various attack scenarios The analysis was verified by using WSInject. A number of vulnerabilities were reduced by the security tool proposed.</p>
6.	Fabien et. al.[35]	IEEE Xplore/2012	<p>Authors presented a methodology in which test inputs are created by combining model inference as well as evolutionary fuzzing methodology. By using this, the web injection vulnerabilities can be detected. GA i.e. Genetic Algorithm is used to generate input. An automated XSS search technique was proposed which uses model inference as well as evolutionary fuzzing methodology for the creation of test cases.</p>
7.	Yu sun et.al.[36]	IEEE Xplore/2012	<p>Authors for defense against cross site scripting attacks proposed a model checking method. Bugs present in the e-commerce website were found and counter examples were shown by model checking. Proposed an automatic modeling algorithm for the HTML code and presented the case of performance of the algorithm.</p>
8.	Lwin Khin et.al. [37]	IEEE Xplore/2012	<p>Authors proposed different input sanitization strategies into various types and proposed a set of static code attributes. They have utilized data mining strategies to anticipate SQLI and XSS vulnerabilities. They have explained classification schemes depending on CFG (Control</p>

			Flow Graph) for web based applications. They have implemented a tool called PhpMiner. It is utilized to extract the information and proposed characteristics from PHP programs.
9.	Lwin Khin et.al. [38]	IEEE Computer Society/2012	Authors proposed various XSS exploits techniques that are similar to SQLI. The attacks are caused due to the improper sanitization of user input. They have also proposed various types of XSS defensive techniques such as detection of web vulnerabilities, prevention of attack at runtime etc.
10.	Avancini et. al. [39]	IEEE Xplore/2011	Authors proposed a search based methodology for security testing of web based applications. They exploited static based analysis to detect XSS vulnerabilities. They have used genetic algorithm based approach. Search based test cases are utilized by developers to resolve security related issues. They have implemented this methodology in a model and tested on PHP based applications.
11.	Rattipong et. al. [40]	IEEE Xplore/2011	Authors presented a methodology for protection of cookies from XSS attack. This method changes cookies in a manner so that the cookies become unworkable for the cross-site script attacks. Prior to the sending of cookie to browser, the cookies are removed with randomized value. This procedure is tested on HTTP connections.
12.	Adam et. al.[41]	IEEE Xplore/2009	Authors created an automatic system which exposes SQLI and XSS vulnerabilities through generating input test cases. In

			<p>this technique, test inputs are created and changed to deliver concrete exploits and track corrupts through execution. The proposed methodology was executed for PHP applications by a tool named Ardilla, which can make input for single script at once but it is not able to simulate sessions(i.e. applications which involve multiple pages user-interaction). Ardilla cannot produce attacks for sink.</p>
13.	Mike et.al.[42]	IEEE Xplore/2009	<p>Authors proposed an XSS defensive methodology, in spite of behaviour anomalous browser it was intended to be effective in existing browser systems. The proposed approach minimizes trust placed on system browser for interpreting non trustable content. BLUEPRINT is a tool used to implement this approach and it was integrated with a few popular web applications. It was a strong way to prevent cross-site scripting attacks which was effective upon 96% of the system browser.</p>
14.	Wassermann et. al. [43]	ACM/2008	<p>Authors presented a static analysis which specifically addresses weak or missing input validation for discovering cross-site scripting vulnerabilities. This approach combines data obtained through corrupted information flow with string analysis. This approach confronts a similar obstacle that statistically check vulnerabilities. This is addressed by formalizing a policy which is given by W3C. An approach to find cross site script web based vulnerabilities because of non-</p>

			checking of malicious information and insufficiently checked entrusted data is proposed in this paper.
15.	Shanmugam et. al. [44]	IEEE Xplore/2007	Authors proposed an approach to detect behavior based anomaly which presents a security layer on top of the websites, due to which whenever new threats appears the mechanisms are changed but the current websites stay unmodified. Also, to reduce processing time application parameters are acquainted, this approach gives security to websites by permitting tags to be entered in the websites. To decrease processing time, it uses whitelist security model due to which it is not prone to zero-day attacks.
16.	Qianjie Zhang et. al.[45]	IEEE Xplore/2010	Authors provided a prevention mechanism which prevents Cross-site Scripting (XSS) attacks using the execution flow mechanism for JavaScript running on client side. Firstly, client side behavior of Ajax applications under normal circumstances is modeled as finite-state-automata(FSA) and deployed in proxy mode. Before running any JavaScript on client side it is analyzed against this model and only on conformation with this model it is allowed to run.
17.	Shanmugam et. al. [46]	IEEE Xplore/2007	Authors proposed a technique which presents a separate security layer above existing web applications which gives flexibility to these application not to be modified but yet to provide sufficient mechanism to prevent any new security threat.
18.	Shanmugam et.	IEEE Xplore/2007	Authors proposed a solution for

	al. [47]		prevention against Cross Site Scripting (XSS) attack which is independent of any particular language implementation and is based on service oriented architecture on which various web applications architecture are based. XSS blocker has independence from any particular platform as well as language because development of interfaces such as converter as well as validator is required to be designed. The task is minimal.
19.	Shiuh-Jeng Wang et. al.[48]	IEEE Computer Society/2007	Authors proposed a scheme that can be used to collect digital evidences on web systems after occurrence of XSS attack which then can be presented in court as evidence and help to recreate the crime-venue in crime case. Stolen personal information is also investigated in this scheme.

3.4 SQL INJECTION ATTACK

SQLI is a standout amongst the most widely recognized major threat to database driven applications security. [5,22] SQL injection is a method where malicious SQL queries are induced as an input so as to exploit the weakness present within database. The attacker tries to inject malicious data. The unauthorized access takes place after the execution of malicious input. It permits a hacker to pick up control over the database of an application and therefore, a hacker will be able to change the data. [23,24,25]

SQL injection attacks are often motivated by illegal activities like monetary gain, fraud, cyber terrorism or even for malware distribution. As per the Whitehat security, around 16% of websites are exposed to SQLIA. About 75 percent of web hacking attacks are launched on shopping carts, login forms and dynamic contents.

3.4.1 Related Work for SQL Injection Attack

- **Sharma et.al.[49]** proposed an integrated method for the mitigation of SQLI and reflected XSS vulnerability. This methodology is using a query based model generator for different types of queries. Queries which are defying the model are prone to attack.
- **Gao Jiao et.al. [50]** presented a mechanism for the prevention of SQLI. This is done by inserting a middleware named SQLIMW which is placed in the background. This paper enforced the concept of authentication by using hash function and a private key along with traditional password. The idea of detection lies in number of queries returned in result set after query execution. In case of, registered user, query set will be not equal to 0 and since username is unique, there will be only one record matching that username, if it exceeds 1, then SQLIMW detects if it is a SQL injection attack or not.
- **Ramya Dharam et. al.[51]** proposed a post deployment monitoring methodology for handling tautology based SQL injection for java based applications. The purpose is to identify critical variables used for accepting inputs from external environment and to identify all valid paths which could be traversed by these critical variables. If the critical variable breaks the checkpoint and follows a path which is invalid, then the monitor identifies the abnormal behaviour and administrator is informed.
- **Indrani Balasundaram et.al.[52]** presented an authentication based scheme which uses hybrid encryption i.e. combination of both Advance Encryption Standard i.e AES as well as Rivest-Shamir-Adleman i.e. RSA. This is used to mitigate SQLI attacks. This methodology uses two stage encryption on login query. A secret key which is unique is issued to every user and server uses combination of private as well as public key for Rivest-Shamir-Adleman i.e. RSA encryption. Asymmetric key encryption via server's public key is used to encrypt

the query. Symmetric key encryption via the secret key is used as encryption of username as well as password.

- **Kunal et. al. [53]** proposed a model based approach i.e MHAPSIA. It is a combination of two phases. During first phase, a model of legitimate queries are constructed. During second phase, it monitors dynamically generated queries. Queries which are defying the model are prone to SQL Injection attack and are prevented from execution.
- **Kai X. Zhang et.al. [54]** presented a methodology named TransSQL. It is a translation-validation solution to detect malicious SQL queries. The proposed scheme duplicates the database into a LDAP database and queries generated by web application is also converted into LDAP queries. These LDAP queries are executed in LDAP database and result is compared with corresponding result from SQL database. In case of mismatch, SQL injection is detected. Once the SQLI is detected, the result is displayed as null result and returned back to application.
- **Allen Pomeroy et.al. [55]** used network recording to reconstruct SQL Injection attack effectively. In order to find vulnerabilities in web applications the authors suggested this technique of network recording. This approach uses NIDS i.e. network intrusion detection system for the network recording of malicious applications.
- **Bisht et. al. [56]** presented an approach named CANDID. It is a tool which is proposed by authors is used for the recording the structure of SQL query given by genuine user. It is then compared by the query structure given by the hacker input.
- **Michelle Ruse et.al. [57]** proposed an approach to detect SQLI by using CREST i.e. automatic test case generation. This framework is built upon the idea of

capturing different parts of the query. It identifies the situation where the vulnerability of queries occurs. The results show that this method is effective.

- **Xin Wang et.al.[58]** presented a web crawling methodology which uses access authorization data table (AADT).By recording authorization information through cookies, session etc. these web crawlers accesses pages which is lying after login forms. To improve the capability of web scanner, the web vulnerability detection mechanism is done. Each and every hidden hyperlink and form present in pages is crawled by crawlers to improve overall web page detection ability of web scanner.
- **Ezumalai et.al.[59]** proposed SQLI detection technique which is signature based. SQL queries in web application are divided into smaller units called tokens which are sent for validation. Hirschberg's algorithm is used for the detection of SQLIA to validate tokens. No runtime changes are required.
- **M. Junjin et. al. [60]** proposed an approach which provides a fully automated system which detects SQL vulnerabilities and hence preventing SQL injection attacks. It automatically generates SQL queries based on legitimate queries. Then all runtime generated queries are checked for their compliance which on any mismatch throws a predefined exception. It has limitation of not having any other implementation except for JSP based web application. This system is named as AMNeSIA.
- **Stephen Thomas et.al. [61]** presented an algorithm to prevent SQL vulnerabilities. For this, the secured prepared statements are used in place of prepared statements for SQL queries. In this approach, the static structure of sql query is changed to logical structure. This approach uses a tool to implement automated fix generation. Based on experimental result, this approach has been able to replace 94% of SQLI vulnerabilities.

- **Kiani et.al. [62]** prepared an SCC model i.e. Same Character Comparison. According to the given paper, author has studied the Foundation Capacities for Development (FCDs) models and removes some limitation to detect attacks by introducing a new model. According to this methodology, the query given in the HTTP request is analyzed and a profile is created for every file. HTTP requests are intercepted by the model. The model extracts the query from HTTP request. In the testing phase, the thresholds are used to find out malicious requests.
- **McClure et.al. [63]** proposed a SQL DOM framework with supporting 'sqldomgen' which implements those classes which are robust to database for SQL statement generation. It is efficient in solving compiler errors like data type mismatch etc. The sqldomgen detects error in code that accesses database along with focusing on identifying obstacles in database interaction via call level interfaces.
- **Valeur et.al. [64]** used a learning based methodology for the detection of SQLI. Here, an IDS is designed which is using machine learning approach. The model is generated by learning SQL queries. Further, discrepancies are checked with the model. There may occur false positive and negative.
- **Gould et. al. [65]** proposed a tool named JDBC checker for SQL/JDBC web applications. This is a static analysis tool. To randomize SQL queries a proxy is lying in between database server as well as web server. This technique uses random values during runtime SQL and tests for the detection of SQLI attacks.
- **Huang et. al. [66]** estimated web security using fault injection, thus monitor sites behaviour. They have designed a tool named Waves which is a web crawler which is used to identify vulnerabilities. It identifies different patterns and methodologies which is used to perform attack and thus generate attack codes. These attack codes are used to identify SQLI. Then, it will generate reports as well with the help of their listed codes which are vulnerable. The method uses machine learning

approach. It is better and effective than other methods which uses penetration testing.

The detailed literature survey of SQL Injection attack can be shown below using Table3.2.

Table3.2. Literature Survey of SQL Injection attack

S. No.	Authors	Publisher/Year	Description
1.	Sharma et.al.[49]	Springer/2012	Authors proposed an integrated method for the mitigation of SQLI and reflected XSS vulnerability. This methodology is using a query based model generator for different queries. Queries which are defying the model are prone to attack
2.	Gao Jiao et.al. [50]	IEEE Xplore/2012	Authors presented a mechanism for the prevention of SQLI. This is done by inserting a middleware named SQLIMW which is placed in the background. This paper enforced the concept of authentication by using hash function and a private key along with traditional password. The idea of detection lies in number of queries returned in result set after query execution.
3.	Ramya Dharam et. al.[51]	IEEE Xplore/2012	Authors proposed a post deployment monitoring methodology for handling tautology based SQL injection for java based applications. The purpose is to identify critical variables used for accepting inputs from external environment and to identify all valid paths which could be traversed by these critical variables.
4.	Indrani Balasundaram	European Journal of Scientific	Authors presented an authentication based scheme

	et.al. [52]	Research/2011	which uses hybrid encryption i.e. combination of both Advance Encryption Standard i.e AES as well as Rivest-Shamir-Adleman i.e. RSA. This is used to mitigate SQLI attacks.This methodology uses two stage encryption on login query. A secret key which is unique is issued to every user and server uses combination of private as well as public key for Rivest-Shamir-Adleman i.e. RSA encryption.
5.	Kunal et. al. [53]	Springer Conference/2011	Authors proposed a model based approach i.e MHAPSIA .It is a combination of two phases. During first phase, a model of legitimate queries are constructed. During second phase, it monitors dynamically generated queries.
6.	Kai X. Zhang et.al. [54]	IEEE Xplore/2011	Authors presented a methodology named TransSQL. It is a translation-validation solution to detect malicious SQL queries. The proposed scheme duplicates the database into a LDAP database and queries generated by web application is also converted into LDAP queries. These LDAP queries are executed in LDAP database and result is compared with corresponding result from SQL database.
7.	Allen Pomeroy et.al. [55]	IEEE Xplore/2011	Authors used network recording to reconstruct SQL Injection attack effectively. In order to find vulnerabilities in web application the authors suggested this technique of network recording. This

			approach uses NIDS i.e. network intrusion detection system for the network recording of malicious applications.
8.	Bisht et. al. [56]	ACM Transactions/2010	Authors presented an approach named CANDID. It is a tool which is proposed by authors is used for the recording the structure of SQL query given by genuine user. It is then compared by the query structure given by the hacker input.
9.	Michelle Ruse et.al. [57]	Symposium/2010	Authors proposed an approach to detect SQLI by using CREST i.e. automatic test case generation. This framework is built upon the idea of capturing different parts of the query. It identifies the situation where the vulnerability of queries occur. The results show that the method is effective.
10.	Xin Wang et.al.[58]	IEEE Xplore/2010	Authors presented a web crawling methodology which uses access authorization data table (AADT).By recording authorization information through cookies, session etc. these web crawlers accesses pages which is lying after login form. To improve the capability of web scanner, the web vulnerability detection mechanism is done.
11.	Ezumalai et.al.[59]	IEEE Xplore/2009	Authors proposed SQLI detection technique which is signature based. SQL queries in web application are divided into smaller units called tokens which are sent for validation. Hirschberg's algorithm is used for the detection of SQLIA to

			validate tokens. No runtime changes are required.
12.	M. Junjin et. al. [60]	Conference/2009	Authors proposed an approach which provides a fully automated system which detects SQL vulnerabilities and hence preventing SQL injection attacks. It automatically generates SQL queries based on legitimate queries. Then generated queries are checked for their compliance. Any mismatch throws a predefined exception.
13.	Stephen Thomas et.al. [61]	ACM/2009	Authors presented an algorithm to prevent SQL vulnerabilities. For this, the secured prepared statements are used in place of prepared statements for SQL queries. In this approach, the static structure of SQL query is changed to logical structure. This approach uses a tool to implement automated fix generation.
14.	Kiani et.al. [62]	IEEE Xplore/2008	Authors prepared an SCC model i.e. Same Character Comparison. According to the given paper, author has studied the Foundation Capacities for Development (FCDs) models and removes some limitation to detect attacks by introducing a new model. According to this methodology, the query given in the HTTP request is analyzed and a profile is created for every file. HTTP requests are intercepted by the model. The model extracts the query from HTTP request.
15.	McClure et.al. [63]	ACM Conference/2005	Authors proposed a SQL DOM framework with

			supporting 'sqldomgen' which implements those classes which are robust to database for SQL statement generation. It is efficient in solving compiler errors like data type mismatch etc. The sqldomgen detects error in code that accesses database along with focusing on identifying obstacles in database interaction via call level interfaces.
16.	Valeur et.al. [64]	ACM Conference/2005	Authors used learning based methodology for the detection of SQLI. Here, an IDS is designed which is using machine learning approach. The model is generated by learning SQL queries. Further, discrepancies are checked with the model. There may occur false positive and negative.
17.	Gould et. al. [65]	ACM Conference/2004	Authors proposed a tool named JDBC checker for SQL/JDBC web applications. This is a static analysis tool. To randomize SQL queries a proxy is lying in between database server as well as web server. This technique uses random values during runtime SQL and tests for the detection of SQLI attacks.
18.	Huang et. al. [66]	ACM Conference/2003	Authors estimated web security using fault injection. They have designed a tool named Waves which is a web crawler which is used to identify vulnerabilities. It identifies different patterns and methodologies which is used to perform attack and thus generate attack codes. These attack codes are used

			to identify SQLI. Then, it will generate reports.
--	--	--	---

3.5 CSRF ATTACK

CSRF attack arises when a non trusted website causes a client's web browser to permit a malicious activity on a trusted website. This is due to the fake HTTP request as it exploits the currently running client's session of the web browser. A CSRF web attack requires inclusion of three things. A target client, a trustable website, and a non trustable web site. The target client is currently holding an active session with a trustable site and in the meanwhile, the client visits a malicious or non trusted website. The non trustable or malicious web site injects a HTTP request for the trustable web site into the target client's session which compromises its integrity. These vulnerabilities permit a hacker to exchange money out from client's account, to collect client's email id, disregard client privacy etc.

3.5.1 Related Work for CSRF Attack

- **Adam Barth et.al.[67]** described new forms of CSRF attack as well as existing CSRF defense techniques and their shortcomings. They contributed a threat model of CSRF based on login activity and network connectivity. They also told about the referrer header for the validation in the browser. In threat model, they have described two types of threats i.e. IN scope threats and Out of scope threats. In the end about the session vulnerabilities and its defenses for cookies and scripting languages.
- **Jovanovic et.al.[68]** proposed a prototype and demonstrated on how to secure the web based open source applications by not disturbing their behavior using experimental results. Prototype is basically about a complete automatic protection from CSRF attacks. It detects and prevents itself without knowing to

the users as well as to the web applications. Their experimental result shows that the solution is useful in protecting vulnerable applications.

- **Mohd. Shadab Siddiqui et.al.[69]** gave an introduction about the CSRF attack and how it is performed with three main things. The three main things are victim user, trusted site and a malicious site. Also, they discussed about how the CSRF attack is different from XSS with different types of vulnerabilities. It also shows different ways to perform CSRF attack like get-post or using an image or script source and limitations of CSRF attack. And in the end, they discussed about the protection against CSRF attack.
- **Pavol Zavorsky et.al.[70]** described OWSAP CSRF guard which is Open Web Application security project to protect against CSRF attacks. They explored how CSRF guard blocks or unblock CSRF attempts through the use of different CSRF models and what are the possible limitations with the CSRF guard after using it. In the end, it shows the possible ways where CSRF guard security work as a mitigation strategy for web applications.
- **Wim Maes et.al.[71]** proposed a client side policy of enforcement framework to protect the users from CSRF transparently. For this, they monitored all outgoing request by web within the browser and used a cross domain policy using their framework. They also proposed a policy for an optimal server side to improve the client side policy. They implemented prototype as a Firefox extension within web 2.0 context.
- **Tatiano Alexenko et.al.[72]** presented how CSRF is a potential threat to the web applications. They provided different ways how the web based applications are exploited. They discussed the already existed countermeasures as well as drawbacks with the proposed solution. The authors suggested the installation of simple extension for notification of cross site request forgery vulnerability as Referer Header is the most common method for defense of CSRF attack.

- **Xialoi Lin et.al.[73]** proposed a threat model and presented a tree based attack analysis of CSRF attacks to help researchers to design defenses for CSRF attacks because different process is used to execute on victim browser. They also discussed major categories of cross site request forgery i.e. reflected and stored. In the end, they mentioned different tree models for CSRF attacks with mitigation over it.
- **Hossain Shahriar et.al.[74]** proposed a mechanism based on detection of cross site request forgery i.e. CSRF vulnerability by checking the request on contents of suspected links. They demonstrated the mechanism by intercepting the request having values which associate this with the form which are noticeable in the windows. If the exact match is found, it will modify the suspected request and transmit this to remote website. It then tries to find the contents and its type. On mismatch, it will normally show a display warning. They implemented a plugin for Firefox browser and tested on different PHP applications.

The detailed literature survey of CSRF attack can be shown below using Table3.3.

Table3.3. Literature Survey of CSRF attack

S. No.	Authors	Publisher/Year	Description
1.	Adam Barth et.al.[67]	ACM Conference/2008	Authors described new forms of CSRF attack as well as existing CSRF defense techniques and their shortcomings. They contributed a threat model of CSRF based on login activity and network connectivity. They also told about the referrer header for the validation in the browser. In threat model, they have described two types of threats i.e. IN scope threats and Out of scope threats.
2.	Jovanovic et.al.[68]	IEEE Xplore/2006	Authors proposed a prototype and

			demonstrated on how to secure the web based open source applications by not disturbing their behavior using experimental results. Prototype is basically about a complete automatic protection from CSRF attacks. It detects and prevents itself without knowing to the users as well as to the web applications.
3.	Mohd. Shadab Siddiqui et.al.[69]	IEEE Xplore/2011	Authors gave an introduction about the CSRF attack and how it is performed with three main things. The three main things are victim user, trusted site and a malicious site. Also, they discussed about how the CSRF attack is different from XSS with different types of vulnerabilities. It also shows different ways to perform CSRF attack like get-post or using an image or script source and limitations of CSRF attack.
4.	Pavol Zavorsky et.al.[70]	IEEE Xplore/2011	Authors described OWSAP CSRF guard which is Open Web Application security project to protect against CSRF attacks. They explored how CSRF guard blocks or unblock CSRF attempts through the use of different CSRF models and what are the possible limitations with the CSRF guard after using it.
5.	Wim Maes et.al.[71]	ACM Workshop/2009	Authors proposed a client side policy of enforcement

			framework to protect the users from CSRF transparently. For this, they monitored all outgoing request by web within the browser and used a cross domain policy using their framework. They also proposed a policy for an optimal server side to improve the client side policy.
6.	Tatiano Alexenko et.al.[72]	IEEE Xplore/2010	Authors presented how CSRF is a potential threat to the web applications. They provided different ways how the web based applications are exploited. They discussed the already existed countermeasures as well as drawbacks with the proposed solution.
7.	Xialoi Lin et.al.[73]	IEEE Xplore/2009	Authors proposed a threat model and presented a tree based attack analysis of CSRF attacks to help researchers to design defenses for CSRF attacks because different processes are used to execute victim browser. They also discussed major categories of cross site request forgery i.e. reflected and stored.
8.	Hossain Shahriar et.al.[74]	IEEE Symposium/2010	Authors proposed a mechanism based on detection of cross site request forgery i.e. CSRF vulnerability by checking the request on contents of suspected links. They demonstrated the mechanism by

			intercepting the request having values which associate this with the form which are noticeable in the windows. If the exact match is found, it will modify the suspected request and transmit this to remote website.
--	--	--	---

3.6 CONCLUSION

A critical study of literature available in the area of web attacks has been performed and some shortcomings were identified in the existing techniques. In order to overcome the drawbacks of the existing techniques available in the literature, a hybrid security system for web attacks is proposed. The proposed approach is discussed in the next chapter in detail.

CHAPTER IV

A HYBRID SECURITY SYSTEM FOR PREVENTION OF XSS, SQL INJECTION AND CSRF WEB ATTACK: PROPOSED APPROACH

4.1 INTRODUCTION

There is a continuous growth of web attacks on web based applications. SQL injection i.e. SQLI, Cross Site Script i.e. XSS, Cross Site Request Forgery i.e. CSRF are some of the most commonly found serious and dangerous threats to the security of web based applications. Hacking permits hacker to pick up access over the database and subsequently, a hacker might have the capacity to change information. The vast majority of the day by day activities rely on database driven web applications as a result of expanding task, such as banking etc. For performing different tasks, for example, paying of bills etc. information should be confidential.

In light of the expanded number of assaults exploiting, many endeavors have been made to discover solution for the issue. The best arrangement is to create the programs in a safe way. Many archives have been distributed in regard to secure advancement of web based applications although very little has managed. Web engineers are not yet security mindful, and the issues keep on appearing. Accordingly, security administrators are continuously searching for different measures that can be taken against this issue. Developers are not yet security aware, and the issues continue to appear. Thus security experts are constantly looking for some other countermeasures which can be considered against the problem.

Although there exist many detection and prevention techniques in the literature, there are certain points where the existing methods can be optimized or there is a requirement of new technique. In order to counter the increased number of attacks taking advantage of the confidential access of information, a security system for the most commonly found serious and dangerous web based attacks is proposed. It is a hybrid system which is developed in PHP. This hybrid security system prevents the most commonly found serious and dangerous web based attacks which are

Cross Site Script i.e. XSS, SQL Injection i.e. SQLI, Cross-Site Request Forgery i.e. CSRF[75] in a more efficient way by reducing the drawbacks of the existing techniques given by different researchers and thereby to improve performance.

4.2 ABSTRACT VIEW OF PROPOSED HYBRID SECURITY SYSTEM

The Security System is a hybrid system[76] which is a combination of three attacks which are Cross Site Script i.e. XSS, SQL Injection i.e. SQLI, Cross-Site Request Forgery i.e. CSRF. It is developed in PHP to prevent the most commonly found serious and dangerous web based attacks namely XSS, SQL Injection and CSRF. This hybrid security system uses combined analysis of both static and dynamic. The proposed system works in different phases which leads to easy design and implementation. The proposed algorithm is divided into two modes. One is static mode and other is dynamic mode.

During Static mode, the following functions occurred:

- Scan PHP web application
- Identify the hotspot
- Run application under safe mode environment with valid test inputs
- Generate model for each identified hotspot.

During Dynamic mode, following functions occurred:

- Capture dynamic query
- Parse it ,Obtain the tokens and generate model
- Match with the static model
- Results and Error report

4.3 OVERALL ARCHITECTURE OF PROPOSED HYBRID SECURITY SYSTEM

The Hybrid Security System[76] is a combination of three attacks which are Cross Site Script i.e. XSS, SQL Injection i.e. SQLI, Cross-Site Request Forgery i.e. CSRF. The architecture of hybrid security system is divided into four phases. It can be shown below using Figure4.1.

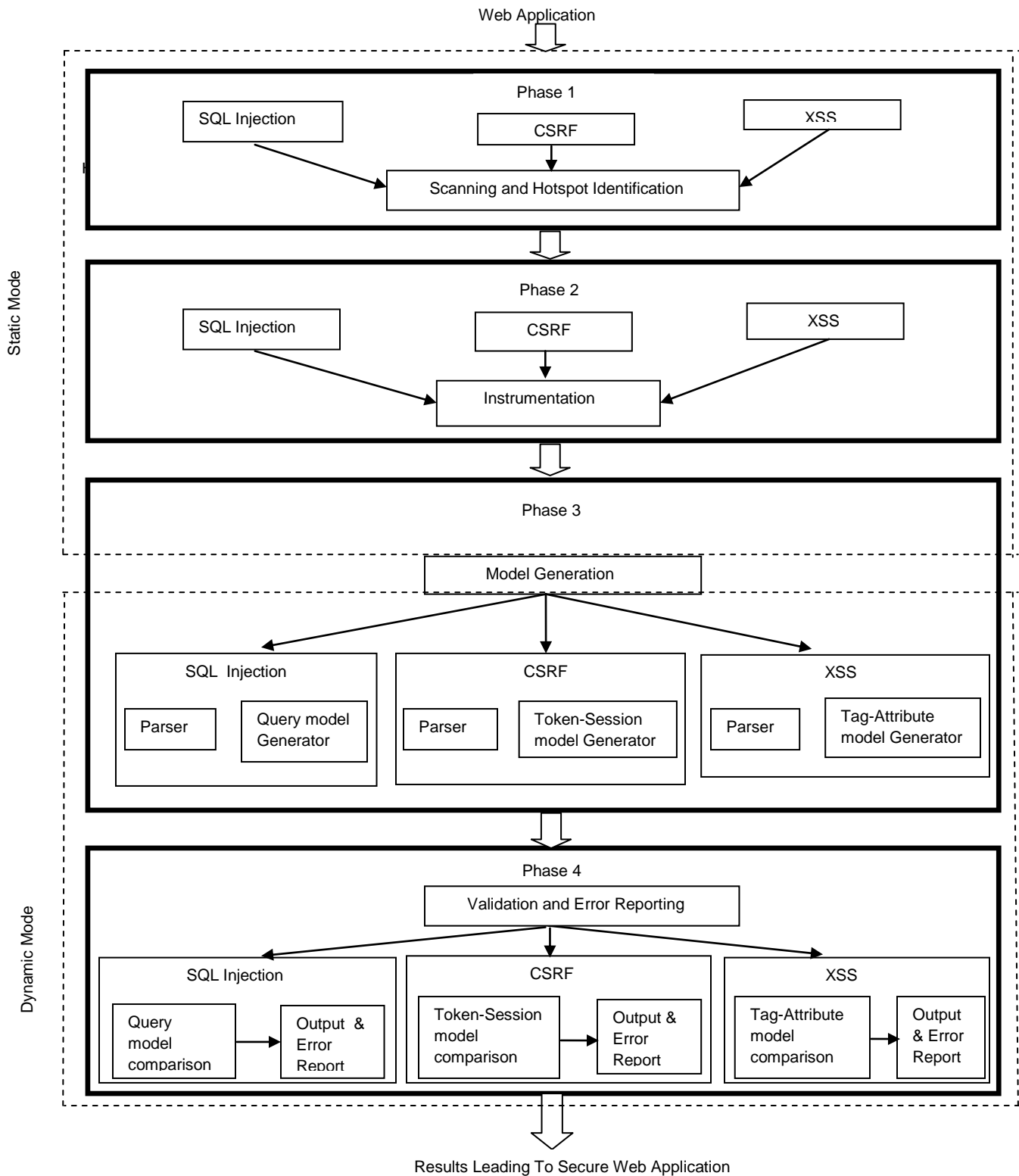


Figure4.1 Architecture of Proposed Hybrid Security System

4.4 PHASES OF HYBRID SECURITY SYSTEM

The Security System is a hybrid system which is developed in PHP to prevent the most commonly found serious and dangerous web attacks which are Cross Site Script i.e. XSS, SQL Injection i.e. SQLI, Cross-Site Request Forgery i.e. CSRF. It is divided into four phases. These four phases are linked with each other. The hybrid system uses combined analysis of both static and dynamic mode. The security system works in different phases which leads to easy design and implementation. As per the proposed algorithm, it is divided into two modes. One is static mode and other is dynamic mode. In static mode, the algorithm statically scans the source code of web application and constructs a static model by considering legitimate input given by the tester/developer while in the dynamic mode it performs verification of queries which are generated during runtime with the model which was generated statically. Queries which violate this model are prone to attack and thus prevented from being executed. The upcoming section discusses different phases of security system.

- **Scanning and Hotspot Identification phase -**

This is the first phase of the security system. This phase scans the web application and identifies hotspot for the most commonly used attacks in the application source code. Hotspot specifies the location of the query to be executed within a web application. These are the places which are used as attack. During this phase, scanning of all the files is done for XSS vulnerable lines within the web application. It returns probable XSS vulnerable lines and termed as hotspot. Likewise, all the files of web application are scanned for SQLI vulnerability. It returns the probable SQLI vulnerable lines and termed as hotspot. At last, all the files of web application are scanned for CSRF vulnerability. It returns the probable CSRF vulnerability and termed as hotspot. The hotspot identification is to spot the number of locations which could be used as an attack in a web application.

- **Instrumentation phase -**

This phase does the Instrumentation of web based application for SQLI, Cross Site Script and CSRF vulnerability. During this phase, at each hotspot an additional code is instrumented which contains a file having runtime checking function at the beginning of file.

- **Model Generation phase -**

This phase does model generation. Three different models have been proposed. First is SQL-Query model which is for SQLI attack. Second is Tag-Attribute model which is for XSS attack. Third is Token-Session model which is for CSRF attack.

The XSS Tag-Attribute model is specifically for XSS attack. In this model, the tokens are generated using parsing of input given at each hotspot. The model is built using this tokenized input. The static mode consists of set of all possible tags and attributes which are vulnerable. The dynamic mode consists of the input at runtime. The model constructed during dynamic mode is compared with the model constructed during static mode. The result of comparison shows the identification of XSS attack or not.

The SQL-Query model is specifically for SQLIA. In this model, the tokens are generated using parsing of query given at each hotspot. The model is built using this tokenized input. It is stored as an array of tokens. The static mode consists of set of all possible legitimate queries. The dynamic mode consists of the input at runtime. The model constructed during dynamic mode is compared with the model constructed during static mode. The result of comparison shows the identification of SQL Injection attack or not.

The Token-Session model is specifically for Cross Site Request Forgery i.e. CSRF. In this model, tokens are generated using parsing of input given at each hotspot. The model is built using this tokenized input. The static mode contains a token id. The dynamic mode consists of the input at runtime. The model constructed during dynamic mode is compared with the model constructed during static mode. The result of comparison shows the identification of CSRF attack or not.

- **Validation and Error Reporting phase -**

This phase does the verification alongwith the error reporting. Validation algorithm performs verification of dynamically generated model in dynamic mode with the statically generated model in static mode. Verification shows the presence or absence of attack. After identification, the attack is prevented using prevention algorithm. Finally, the results and the error report is displayed.

4.5 PREVENTION OF XSS ATTACK USING HYBRID SECURITY SYSTEM

XSS attack is prevented using hybrid security system. The hybrid system uses combined analysis of both static and dynamic mode. This system works in four phases as mentioned below.

4.5.1 Scanning and Hotspot Identification phase

This phase does the scanning of the web application and identifies hotspot for XSS attack in the application source code. During this phase, scanning of all the files is done for XSS vulnerable lines within the web application. It returns probable XSS vulnerable lines and termed as hotspot. Hotspots are the location where actual query gets executed. These are the places which are used as an attack. HTML tags are the primary hotspots in web application e.g. div, heading etc. Hotspot identification is to identify the number of locations which could be used as an attack in a web application.

4.5.2 Instrumentation phase

This phase does the instrumentation of web application. During this phase, at each hotspot an extra code is instrumented at the beginning which contains a file having runtime checking function. The runtime checking function consists of two arguments i.e. one is the string having the script and other is the hotspot id which is unique.

4.5.3 Tag-Attribute Model Phase

The XSS Tag-Attribute model[76] is specifically for XSS attack. In this model, the tokens are generated using parsing of input given at each hotspot. The model is built using this tokenized input. The static mode consists of set of all possible tags and attributes which are vulnerable. The model generated for static mode is shown below in Table 4.1.

Table4.1.Tag-Attribute model for static mode

Tag	Attribute	Example
Script	Src	<script>alert('Cross site script i.e. xss')</script>
Img	Src	<imgsrc="javascript:alert('cross site script i.e. xss')">
Iframe	Src	<iframe src="javascript:alert('cross site script i.e.xss')"></iframe>
Object	Data	<object data="javascript:alert('cross site script i.e.xss')"></object>
Frame	Onmouseclick	<frame onmouseclick="javascript:alert('cross site script i.e.xss')"> </frame>
Frame	Onmouseover	<frame onmouseover="javascript:alert('cross site script i.e.xss')"> </frame>
Frame	Onmouseout	<frame onmouseout="javascript:alert('xss')"></frame>
Div	Style	<div style="javascript:alert('cross site script i.e. xss')"></div>

During Dynamic mode, the tag-attribute model represents the input entered on the login page by the user during run time. For example:

User name - <script>alert('Cross Site Script')</script>

Password - *****

The extracted values for username and password are ” <script>alert('Cross site script i.e. XSS')</script>” and “*****” respectively. These values are stored during dynamic mode as tag attribute model. The model constructed during dynamic mode is compared with the model constructed during static mode. The result of comparison shows the identification of XSS attack or not.

4.5.4 Validation and Error Report Phase

This phase does the verification alongwith the error reporting. Validation algorithm performs verification of dynamically generated model in dynamic mode with the statically generated model in static mode. Verification shows the presence or absence of attack. After identification, the attack is prevented using prevention algorithm. The prevention algorithm checks for the presence of '<' or '>' characters in the input which are an indication of XSS attack. If any of the suspicious characters are found in the input, the algorithm checks the presence of any of the blacklisted tags. Input is tested recursively for presence of blacklisted tags so that attack can be filtered. Whenever presence of blacklisted tag is confirmed in the input, a flag is raised indicating presence of XSS attack. Attack is further filtered for persistent or non-persistent XSS attack by determining whether other users could see this input in their browsers. If input is to be stored on server after which it could be seen by anyone then attack is raised as stored XSS attack otherwise reflected XSS attack. Finally, the results and the error report is displayed.

Its working can be shown below using Figure 4.2.

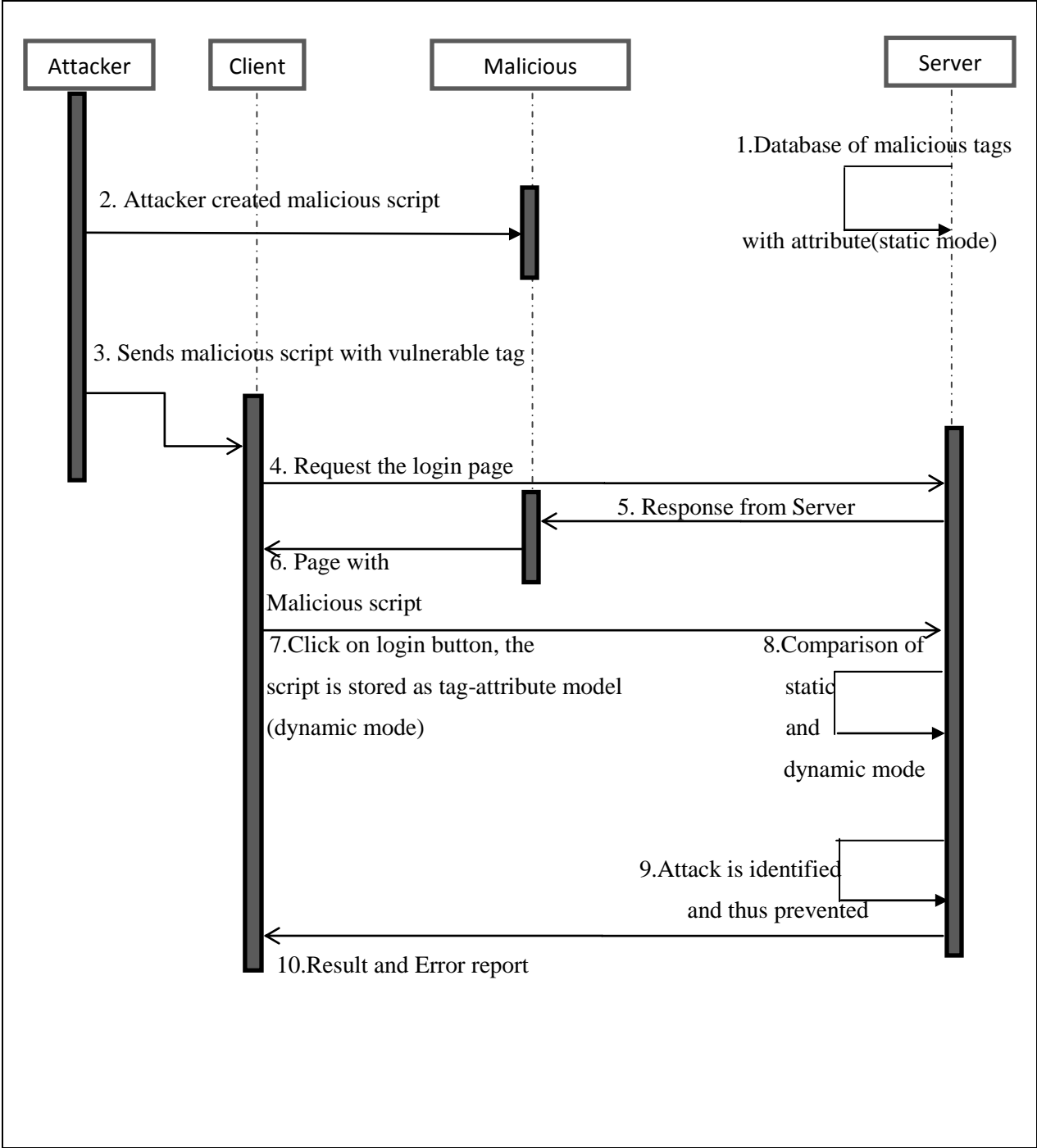


Figure4.2 Prevention of XSS attack

4.6 PREVENTION OF SQL INJECTION ATTACK USING HYBRID SECURITY SYSTEM

SQL Injection attack is prevented using hybrid security system. The hybrid system uses combined analysis of both static and dynamic mode. This system works in four phases as mentioned below.

4.6.1 Scanning and Hotspot Identification phase

This phase does the scanning of the web application and identifies hotspot for SQLI attack in the application. During this phase, all the files of web application are scanned for SQLI vulnerability. It returns the probable SQLI vulnerable lines and termed as hotspots. Hotspots are the location where actual query gets executed. These are the places which are used as an attack. SQL queries are the primary hotspots in web application. Hotspot identification is to identify the number of locations which could be used as an attack in a web application.

4.6.2 Instrumentation phase

This phase does the instrumentation of web application. During this phase, at each hotspot an extra code is instrumented at the beginning which contains a file having runtime checking function. The runtime checking function consists of two arguments i.e. one is the string having the script and other is the hotspot id which is unique.

4.6.3 SQL-Query Model Phase

The SQL-Query model is specifically for SQLIA. In this model, the tokens are generated using parsing of query given at each hotspot. The model is built using this tokenized input. It is stored as an array of tokens. The static mode consists of set of all possible legitimate queries. The model generated for static mode stores SQL query in the form of array. For example, consider the following SQL query during static mode-

```
SELECT * FROM ACCOUNT WHERE USERNAME='alice' AND PASSWORD='abcd';
```

The above mentioned SQL query is stored in an array as SQL-query model during static mode. It can be shown below using Figure 4.3.

```
Array([0]=>SELECT[1]=>*[2]=>FROM[3]=>ACCOUNT[4]=>WHERE[5]=>USERNAME[6]
=>=[7]=> alice[8]=> AND[9]=> PASSWORD[10]=>=[11]=>abcd)
```

Figure4.3. SQL-query model during static mode

Consider the following types of SQL Injection during run time-

i) Tautology based SQL injection-

Let us consider the following example-

```
SELECT * FROM ACCOUNT WHERE USERNAME='alice' OR '1'='1'# AND
PASSWORD='abcd';
```

The above mentioned SQL query is stored in an array as SQL-query model during dynamic mode. It can be shown below using Figure 4.4.

```
Array([0]=>SELECT[1]=>*[2]=>FROM[3]=>ACCOUNT[4]=>WHERE[5]=>USERNAME[6]
=>=[7]=>alice[8]=>OR[9]=>1[10]=>=[11]=>1[12]=>#[13]=>AND[14]=>PASSWORD[15]=>=
[16]=>abcd)
```

Figure4.4. Tautology based SQL-query model during dynamic mode

ii) Union query based SQL injection-

Let us consider following example-

```
SELECT * FROM ACCOUNT WHERE USERNAME='alice' UNION SELECT * FROM
ACCOUNT AND PASSWORD= 'abcd';
```


The above mentioned SQL query is stored in an array as SQL-query model during dynamic mode. It can be shown below using Figure 4.5.

```
Array([0]=>SELECT[1]=>*[2]=>FROM[3]=>ACCOUNT[4]=>WHERE[5]=>USERNAME[6]
=>=[7]=>alice[8]=>UNION[9]=> SELECT[10]=>*[11]=>FROM[12]=> ACCOUNT[13]=>
AND[14]=> PASSWORD[15] =>=[16]=>abcd)
```

Figure4.5. Union query based SQL-query model during dynamic mode

iii) Stored procedure based SQL injection-

Let us consider following example -

```
SELECT * FROM ACCOUNT WHERE USERNAME='alice'; SHUTDOWN AND
PASSWORD='xyz';
```

The above mentioned SQL query is stored in an array as SQL-query model during dynamic mode. It can be shown below using Figure 4.6.

```
Array([0]=>SELECT[1]=>*[2]=>FROM[3]=>ACCOUNT[4]=>WHERE[5]=>USERNAME[6]
=>=[7]=>alice[8]=>;[9]=>SHUTDOWN[10]=>AND[11]=>PASSWORD[12]=>=[13]=>xyz)
```

Figure4.6. Stored Procedure based SQL-query model during dynamic mode

iv) Blind Injection based SQL injection-

Let us consider following example -

```
SELECT * FROM ACCOUNT WHERE USERNAME='alice' AND 1=0 AND
PASSWORD='xyz'
```

The above mentioned SQL query is stored in an array as SQL-query model during dynamic mode. It can be shown below using Figure 4.7.

```
Array([0]=>SELECT[1]=>*[2]=>FROM[3]=>ACCOUNT[4]=>WHERE[5]=>USERNAME[6]
=>=[7]=>alice[8]=>AND[9]=>1=0[10]=>AND[11]=>PASSWORD[12]=>=[13]=>xyz)
```

Figure4.7. Blind Injection based SQL-query model during dynamic mode

v) Piggy-backed query based SQL injection-

Let us consider following example -

```
SELECT * FROM ACCOUNT WHERE USERNAME='alice'; DROP TABLE ACCOUNT
AND PASSWORD='abcd'
```

The above mentioned SQL query is stored in an array as SQL-query model during dynamic mode. It can be shown below using Figure 4.8.

```
Array([0]=>SELECT[1]=>*[2]=>FROM[3]=>ACCOUNT[4]=>WHERE[5]=>USERNAME[6]
=>=[7]=>alice[8]=>;[9]=>DROP[10]=>TABLE[11]=>ACCOUNT[12]=>AND[13]=>PASSWO
RD[14] =>=[15]=>abcd)
```

Figure4.8. Piggy-backed query based SQL-query model during dynamic mode

The dynamic mode consists of the input at runtime. The model constructed during dynamic mode is compared with the model constructed during static mode. This is performed by comparing the length of array calculated during the static mode and the length of array calculated during dynamic mode. The result of comparison shows the identification of SQLI attack or not.

4.6.4 Validation and Error Report phase

This phase does the verification alongwith the error reporting. Validation algorithm performs verification of dynamically generated model in dynamic mode with the statically generated model in static mode. Verification shows the presence or absence of attack. After identification, the attack is prevented using prevention algorithm. The prevention algorithm first extracts input

entered by the user in the form of username and password and stores it in a dummy table in the encrypted mode. The username entered by the user and username that was stored in dummy table will not match so it will give false result. The dummy table will only give true result whenever the username and password are injected as SQL injection query. If the result produced by the dummy table is false, then the username and password given by the user will be forwarded to the actual database and it will show the required result to the user otherwise if the result produced by the dummy table is true, then the username and password given by the user will not be forwarded to the actual database. An error message is generated.

Its working can be shown below using Figure 4.9.

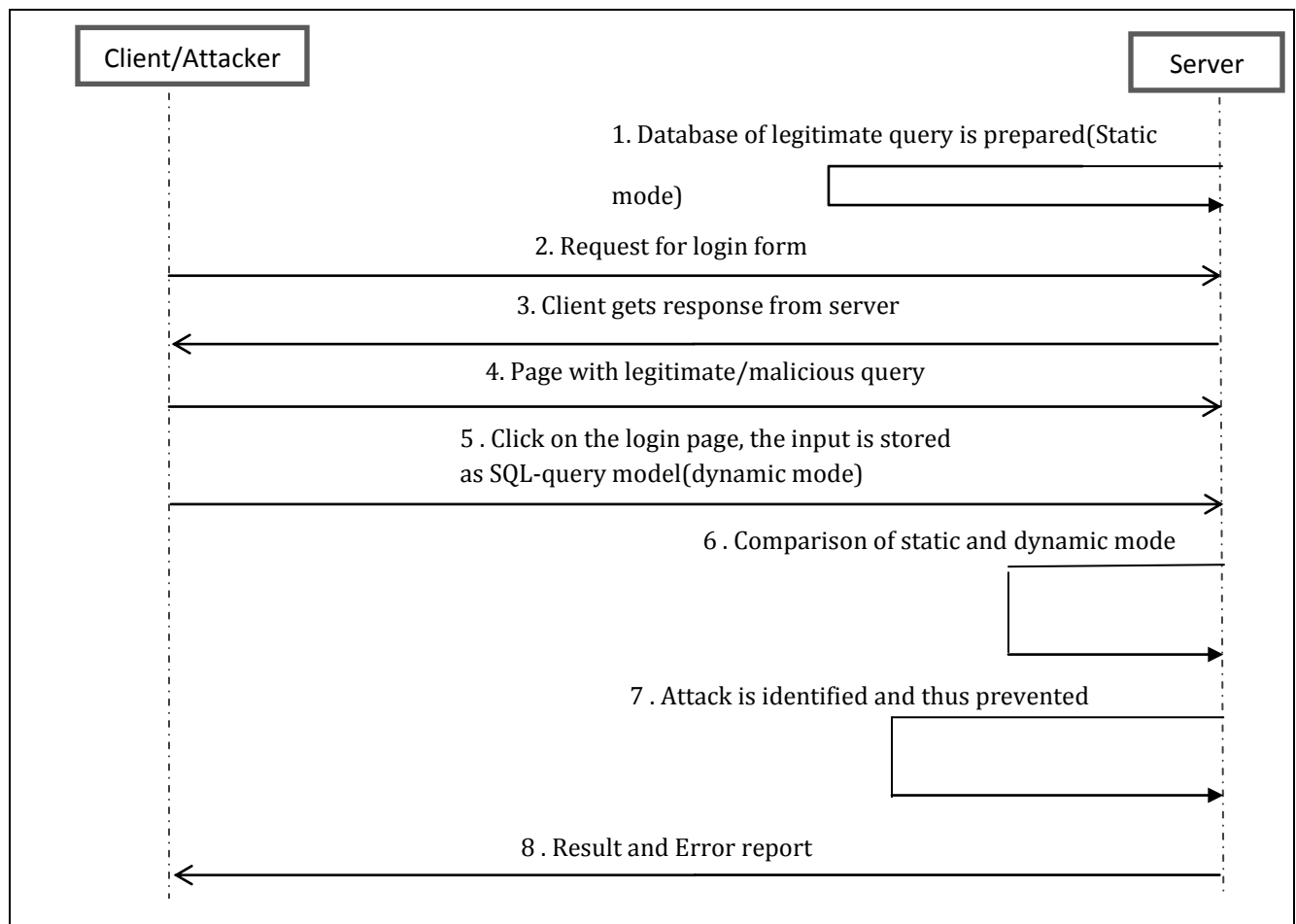


Figure4.9 Prevention of SQL Injection attack

4.7 PREVENTION OF CSRF ATTACK USING HYBRID SECURITY SYSTEM

CSRF attack is prevented using hybrid security system. The hybrid security system uses combined analysis of both static and dynamic mode. This system works in four phases as mentioned below.

4.7.1 Scanning and Hotspot Identification phase

This phase does the scanning of the web application and identifies hotspot for CSRF attack in the application source code. During this phase, all the files of web application are scanned for CSRF vulnerability. It returns the probable CSRF vulnerability and termed as hotspots. Hotspots are the location where actual query gets executed. These are the places which are used as an attack. Hotspot identification is to identify the number of locations which could be used as an attack in a web application.

4.7.2 Instrumentation phase

This phase does the instrumentation of web application. During this phase, at each hotspot an extra code is instrumented at the beginning which contains a file having runtime checking function. The runtime checking function consists of two arguments i.e. one is the string having the script and other is the hotspot id which is unique.

4.7.3 Token-Session Model Phase

The Token-Session model is specifically for CSRF. In this model, the tokens are generated using parsing of input given at each hotspot. The model is built using this tokenized input. User is authenticated with its username and password and allotted a token id from server. The static mode contains a token id. Token id is generated by that particular web page and is unique to web page and user. A static model is created in which each request contains token id which is unique per user and per request. Every user has their unique token id generated by the server and that token id is locked with the IP address provided by the server. The token id is generated once for each session. During dynamic mode, when a forged request for an action is carried out, the attacker may send the script tag to the victim/user to interact with the link and open the same

session which is opened by the victim/user and steal the personal information of it. But in this situation the attacker has its own web browser which has its own token id to interact the same session which is opened by the victim/user. The model constructed during dynamic mode is compared with the model constructed during static mode. The result of comparison shows the identification of CSRF attack or not.

4.7.4 Validation and Error Report phase

This phase does the verification alongwith the error reporting. Validation algorithm performs verification of dynamically generated model in dynamic mode with the statically generated model during static mode. Verification shows the presence or absence of attack. After identification, the attack is prevented using prevention algorithm. The prevention algorithm first requests server for a particular action from a trusted browser. On server side, a token id is extracted from HTTP request. If token id is different, declare CSRF attack and abandon request. Compare the generated token id and the existing token id. If both are not equal, it means given token id is invalid. If given token id is valid then execute action of request, otherwise abandon request. Accordingly, an error report is generated.

Its working can be shown below using Figure 4.10.

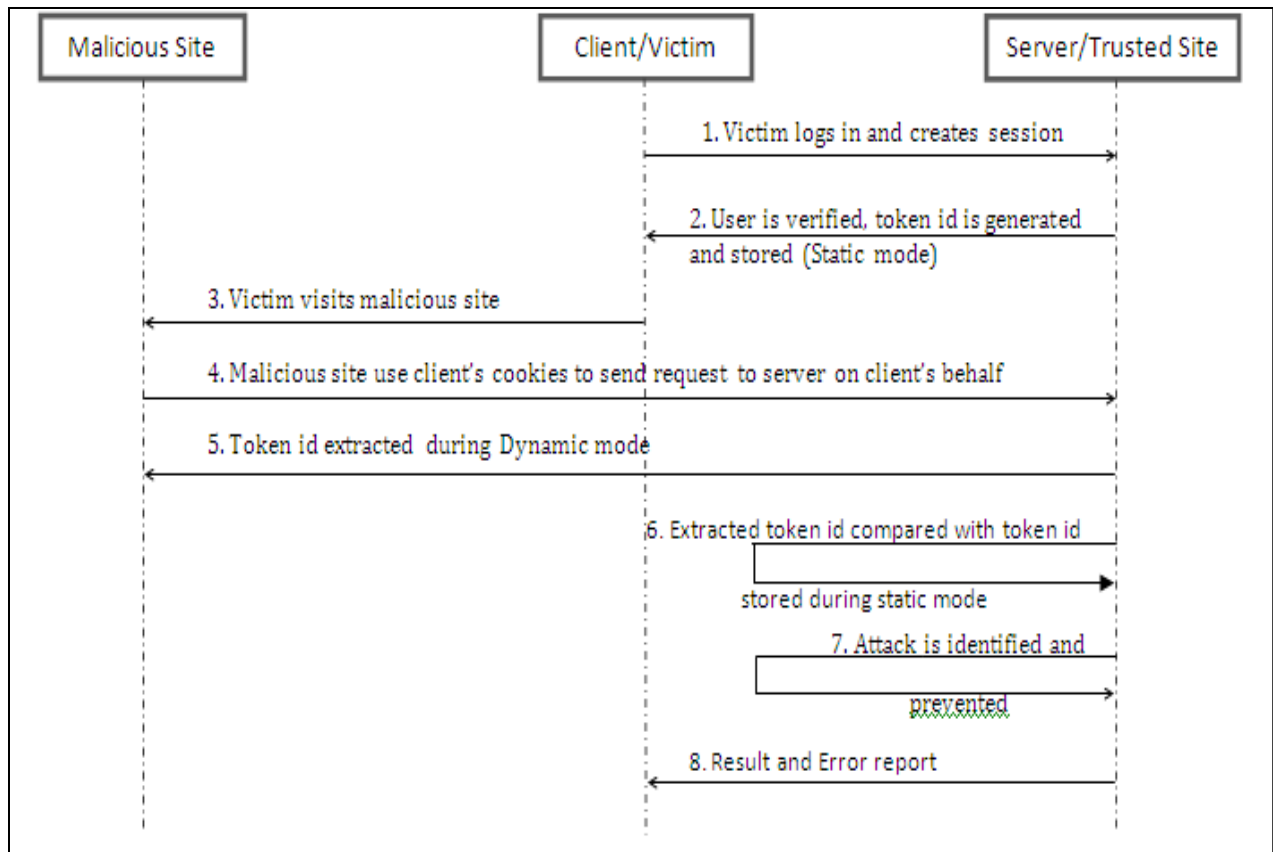


Figure4.10 Prevention of CSRF attack

4.8 ALGORITHMS

The proposed algorithm has been divided into four distinct parts. First part of it focuses on the scanning of web application and identification of hotspots for Cross Site Script i.e. XSS vulnerability, SQLI based vulnerable lines and CSRF based vulnerable lines. Second part of it focuses on the instrumentation of web application. An extra code is attached at every hotspot. It includes file containing dynamic checking function at the beginning of every PHP file. The third part of it focuses on the model generation for XSS, SQL and CSRF respectively. Separate models are generated for XSS, SQL and CSRF attack. Final part of it focuses on validation and error reporting. Validation is done by comparing the models. Depending upon the comparison, error report is generated which identifies the presence of attack.

4.8.1 Algorithm 1: Scanning and Hotspot Identification

The Scanning and Hotspot identification algorithm for SQL Injection, CSRF and XSS attack is shown below in Figure 4.11.

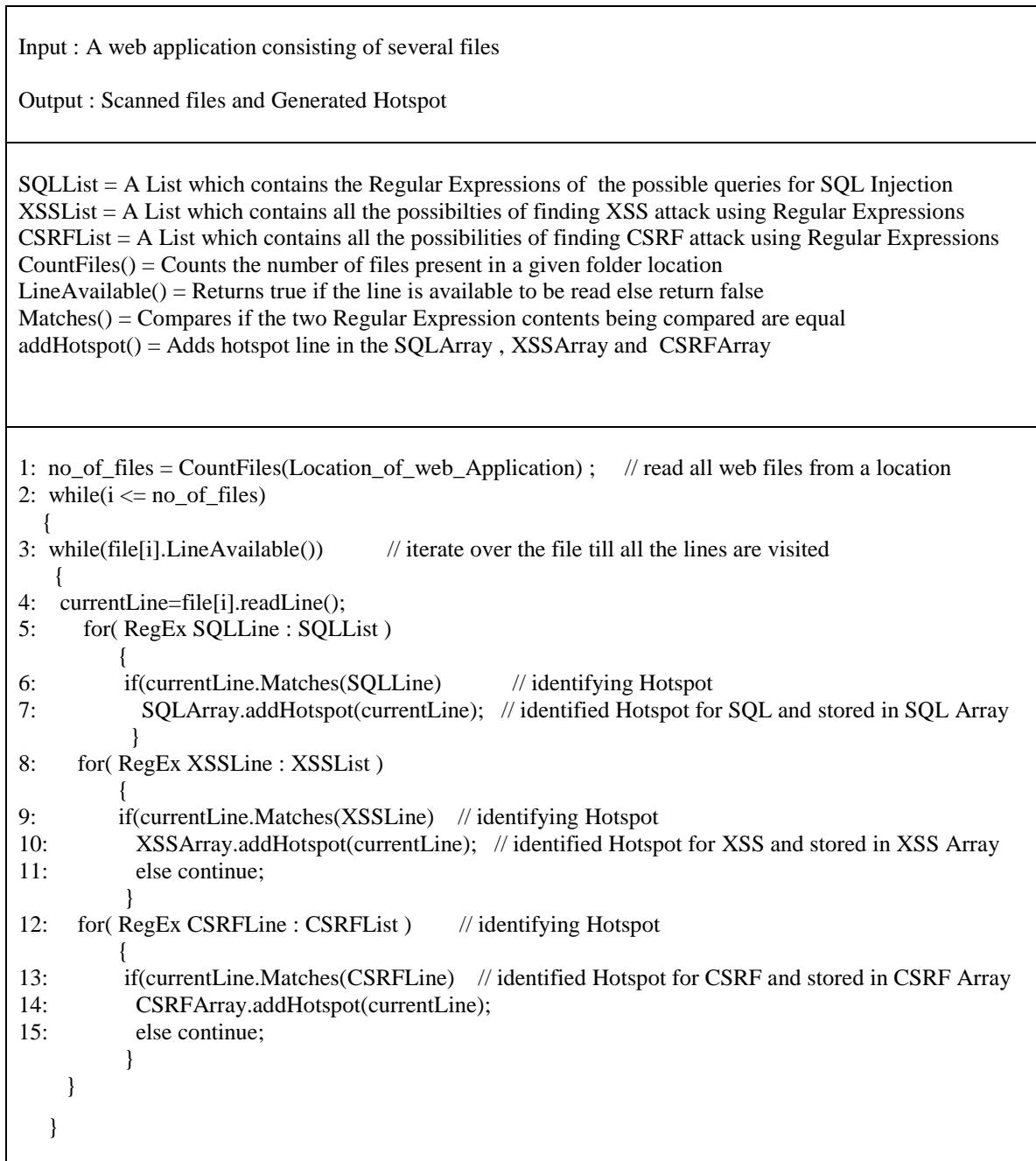


Figure4.11.Scanning and Hotspot identification for SQLI,CSRF and XSS

4.8.2 Algorithm 2: Instrumentation

The Instrumentation algorithm for SQL Injection, CSRF and XSS attack is shown below in Figure 4.12.

<p>Input : Input web pages alongwith Hotspot Identifiers found in the webpage Output : A fresh file with altered changes on which further modeling is to be performed</p>
<p>SQLIHotspotList = A List of Hotspots identified in the scanning and identification of SQL Injection based vulnerable lines in the page XSSHotspotList = A List of Hotspots identified in the scanning and identification of XSS based vulnerable lines in the page CSRFHotspotList = A List of Hotspots identified in the scanning and identification of CSRF based vulnerable lines in the page isAvailableNext()= Checks whether the next hotspot is available for the present input PHP file</p>
<pre>1: void instrumentInputFile(ArrayList SQLIHotspotList,ArrayList XSSHotspotList,ArrayList CSRFHotspotList, File inputFile) { 2: while(SQLIHotspotList.isAvailableNext()) //for SQL Injection { 3: String currentLine = inputFile.searchFor(SQLIHotspotList.next()); 4: currentLine.addInstrumentedLinesSQLI(); // append calls for filtering of tokens. 5: inputFile.saveModifiedFile(); } 6: while(XSSHotspotList.isAvailableNext()) //for XSS { 7: String currentLine = inputFile.searchFor(XSSHotspotList.next()); 8: currentLine.addInstrumentedLinesXSS(); // append calls for filtering of input to check tags and attributes. 9: inputFile.saveModifiedFile(); } 10: while(CSRFHotspotList.isAvailableNext()) //for CSRF { 11: String currentLine = inputFile.searchFor(CSRFHotspotList.next()); 12: currentLine.addInstrumentedLinesCSRF(); // append calls for adding hidden input equal to Session ID. 13: inputFile.saveModifiedFile(); } }</pre>

Figure4.12.Instrumentation for SQL Injection, XSS and CSRF attack

4.8.3 Algorithm 3: Model Generation

The Model Generation algorithm for SQL Injection, CSRF and XSS attack is shown below in Figure 4.13.

<p>Input : String identified as hotspot alongwith the filename Output : Comparison of Static and Dynamic mode</p>
<p>scannedLine = Vulnerable lines found during scanning of the code pageHotspot = The name of the page which is currently being scanned hotspotCount=The number of hotspots identified in a PHP file</p>
<p>SQL Injection Attack</p>
<pre>1:int StaticModeAnalysisSQLI(String scannedLine, File pageHotspot) //performing analysis in Static-Mode for SQLIA { 2: String inputLine = scannedLine.substring(scannedLine.indexOf('=')); 3: String[] line = getStaticModeInput(); 4: String evaluation = performEvaluation(line); // Performs evaluation of the SQL statement on given inputs 5: String[] countStaticMode = evaluation.split("="); 6: int staticModeLength = countStaticMode.length(); 7: return staticModeLength; } 8:int dynamicModeAnalysisSQLI(String scannedLine, File pageHotspot) //performing analysis in Dynamic-Mode for SQLIA { 9: String inputLine = scannedLine.substring(scannedLine.indexOf('=')); 10: String[] line = getdynamicModeInput(); 11: String evaluation = performEvaluation(line); 12: String[] countDynamicMode = evaluation.split("="); 13: int dynamicModeLength = countDynamicMode.length(); 14: return dynamicModeLength; } 15:String compareDifferentModesSQLI(String pageHotspot) //comparison of Static-Mode and Dynamic- Mode for SQLIA { 16:for(int i=0; i< hotspotCount;i++) { 17: int staticModeLength = staticModeAnalysisSQLI(hotspotLine[i],currentPage);</pre>

```

18: int dynamicModeLength =dynamicModeAnalysisSQLI(hotspotLine[i],currentpage);
19:     if(staticModeLength >dynamicModeLength)
20:         return " Tried SQL-Injection Attack in this page " + pageHotspot;
21:     else if(staticModeLength == dynamicModeLength)
22:         return " Operation is safe to perform! No Attack from this page " + pageHotspot;
23:     else
24:         return "Something invalid happened. Please perform operations again! ";
    }
}

```

XSS Attack

1:List staticModeAnalysisXSS(String scannedLine,String pageHotspot) // performing analysis in Static-Mode for XSS

```

{
2: String inputLine = receiveInput();
3: String[] inputSplit = inputLine.split(VulnerableTagList.allTags());
4: for (String atATime : inputSplit)
5:     XSSSafeList.add(atATime);
6: return XSSSafeList;
}

```

7:List dynamicModeAnalysisXSS(String scannedLine,String pageHotspot) // performing analysis in Dynamic-Mode for XSS

```

{
8: String inputLine = receiveInput();
9: String[] inputSplit = inputLine.split(VulnerableTagList.allTags());
10: for (String atATime : inputSplit)
11:     XSSVulnerableList.add(atATime);
12: return XSSVulnerableList;
}

```

13:String compareDifferentModesXSS(String pageHotspot) // comparison of Static-Mode and Dynamic-Mode for XSS

```

{
14: XSSSafeList = staticModeAnalysisXSS(scannedLine,currentPage);
15: XSSVulnerableList = dynamicModeAnalysisXSS(scannedLine,currentPage);
16: if(XSSSafeList.contains(XSSVulnerableTagList.anyTag())
17:     return "Safe Mode result has encountered some problem,please perform the safe again! ";
18: if(XSSVulnerableList.contains(XSSVulnerableTagList.anyTag())
{
19:     return "Vulnerable XSS Attack was tried!!! It has been blocked... ";
20:     FilteredInput = XSSVulnerableList.XSSSanitiser();
21:     displayOutput(FilteredInput);
}
22: if(XSSSafeList.equalElements(XSSVulnerableList))
    return "Operation successfully performed. No threats detected ";
23: else
24:     return "Some error occurred somewhere in between. Please perform this operation again! ";
}

```

CSRF Attack

```
1: EncryptedValue staticModeAnalysisCSRF(String scannedForm,String pageHotspot) // performing
                                     analysis in Static-Mode for CSRF Attack
    {
2: String action = getSessionID(); // performing encryption of Session-ID and storing that in token
3: EncryptedValue smev = action.encrypt();
4: return smev;
    }

5: EncryptedValue dynamicModeAnalysisCSRF(String scannedForm,String pageHotspot) // performing
analysis in Dynamic- Mode for CSRF Attack
    {
6: EncryptedValue real_action = getHiddenInputValueOfFormTriggered();
7: return real_action;
    }

8: String compareDifferentModesCSRF(String scannedForm,String pageHotspot)
    {
9: EncryptedValue smev = staticModeAnalysisCSRF(scannedForm,pageHotspot);
10: EncryptedValue rmev = dynamicModeAnalysisCSRF(scannedForm,pageHotspot); // perform checking
of the tokens at runtime
11: if(smev == null OR rmev = null)
12:     return "Some error occurred while processing information about CSRF attack in page
"+pageHotspot;
13: if(smev.matches(rmev))
14:     return "User is genuine and there doesn't exist any attempt to attack in page "+pageHotspot;
15: else
16:     return "There was an attempt of CSRF attack on the page "+pageHotspot;
    }
```

Figure4.13.Model Generation for SQL Injection, XSS and CSRF attack

4.8.4 Algorithm 4: Prevention of XSS attack

The Prevention algorithm for XSS attack is shown below in Figure 4.14.

```
String Q = username; // Q = username
Int Length = length of Q;
Bool Attack = false;
Int Counter = 1;
Char c;
while (Attack=false or Counter <=Length){
    c = scan nextCharacter(Q); // Scans next character from string
```

```

        Counter++;
        If (c='<' or c='>')
            Attack=checkForTag(Q);
    } // Explained below
If (Attack=true){
    If (input to be stored in public section)
        Print "Stored xss attack";
    Else
        Print "Reflected xss attack";
}
Else
    Print "safe input";

// Function checkForTag(Q)

Char c;
String tag;
Bool attack = false;
While (c!='>' or attack=false){
    c =nextCharacter(Q);
    If (c='<')
        attack=checkForTag(Q);
    Append c to tag; // Appends char c to string tag
Attack = checkTagPresent(tag);
}
return value of attack;

```

Figure4.14. Prevention of XSS attack

4.8.5 Algorithm 5: Prevention of SQL Injection attack

The Prevention algorithm for SQL Injection attack is shown below in Figure 4.15.

```

String input = User input from query string;
Send these inputs to dummy table;
If (exception)
    Redirects user to login page showing an error;
Else
    Check username and password in database;

```

Figure4.15. Prevention of SQL Injection attack

4.8.6 Algorithm 6: Prevention of CSRF attack

The Prevention algorithm for CSRF attack is shown below in Figure4.16.

```
String credentials = user credentials;    // from HTTP request
Int id = token_id of credentials;
Int Session_id = session_id of credentials;
Int Generated_id = Generate(credentials) ; // returns token_id from credentials
If (Token_id = null or Token_id != Generated_id)
    Abandon request;
User is authenticated and requested action is executed;
```

Figure4.16. Prevention of CSRF attack

4.8.7 Algorithm 7: Validation and Error reporting

The Validation and Error reporting algorithm is shown below in Figure4.17.

```
Input : The current input page
Output : The output screen with results and errors

1: void resultObtainedAfterAnalysis() // To perform test and generate errors
   {
2:   String attackSQL = compareDifferentModesSQLI(currentPage);
3:   String attackXSS = compareDifferentModesXSS(currentPage);
4:   String attackCSRF =compareDifferentModesCSRF(currentForm,currentPage);
5:   printOutput(attackSQL);
6:   printOutput(attackXSS);
7:   printOutput(attackCSRF);
8:   printOutput("All Operations performed on the current input page ");
   }
```

Figure4.17.Validation and Error report

4.9 CONCLUSION

The chapter began by justifying the need of security system. It is a hybrid system which is a combination of three attacks which are Cross Site Script i.e. XSS, SQL Injection i.e. SQLI, Cross-Site Request Forgery i.e. CSRF. It is developed in PHP. This proposed hybrid security system prevents the most commonly found serious and dangerous website attacks i.e. XSS, SQL injection and CSRF attack in a more efficient way by reducing the drawbacks of the existing techniques which are being observed and thereby to improve performance. The proposed hybrid security system comprised of four phases which has been explained in this chapter. The implementation and experimental analysis of hybrid security system is discussed in the next chapter in detail.

CHAPTER V

IMPLEMENTATION AND EXPERIMENTAL ANALYSIS

5.1 INTRODUCTION

In the previous chapter, we proposed a hybrid security system which is a combination of three attacks which are SQL Injection i.e. SQLI, Cross Site Script i.e. XSS, Cross Site Request Forgery i.e. CSRF. It is developed in PHP. This security system prevents the most commonly found serious and dangerous web attacks which are XSS, SQLI and CSRF attack in a more efficient way by reducing the drawbacks of the existing techniques given by different researchers which are being observed and thereby to improve performance. The proposed hybrid system works in different phases which leads to easy design and implementation.

5.2 IMPLEMENTATION OF PROPOSED HYBRID SECURITY SYSTEM


The Security System[76] is a tool which is developed in PHP to prevent SQLI,XSS and CSRF attack. The proposed technique is implemented for applications which are PHP based. The tool works in different phases which leads to easy design and implementation.

5.2.1 SQL Injection Vulnerability

SQLI is a standout amongst the most widely recognized major threat to database driven applications security.[5,22] SQL injection is a method where malicious SQL queries are induced as an input so as to exploit the weakness present within database. The attacker tries to inject malicious data. The unauthorized access takes place after the execution of malicious input. It permits a hacker to pick up control over the database of an application and therefore, a hacker will be able to change the data.[23,24,25]

5.2.1.1 Exploiting SQL Injection Vulnerability

The user login page for inputting username and password is shown below using Figure 5.1.



The image shows a user login page titled "User Login Page". It features a central white box with a blue header that says "Login Details...". Below the header, there are two input fields: "Username:" followed by a text box, and "Password:" followed by a text box. Below these fields is a grey "Submit" button.

Figure5.1. User Login Input Page

The user login page to input username and password for legitimate user is shown using Figure5.2.

INPUT:

User Login Page

Login Details...

Username:

Password:

Figure5.2. User Login Page with Legitimate Input

OUTPUT:

The output showing successful login as shown below in Figure5.3.



Figure5.3. Output showing successful login

The User Login Page with malicious input is shown below using Figure5.4.

INPUT:

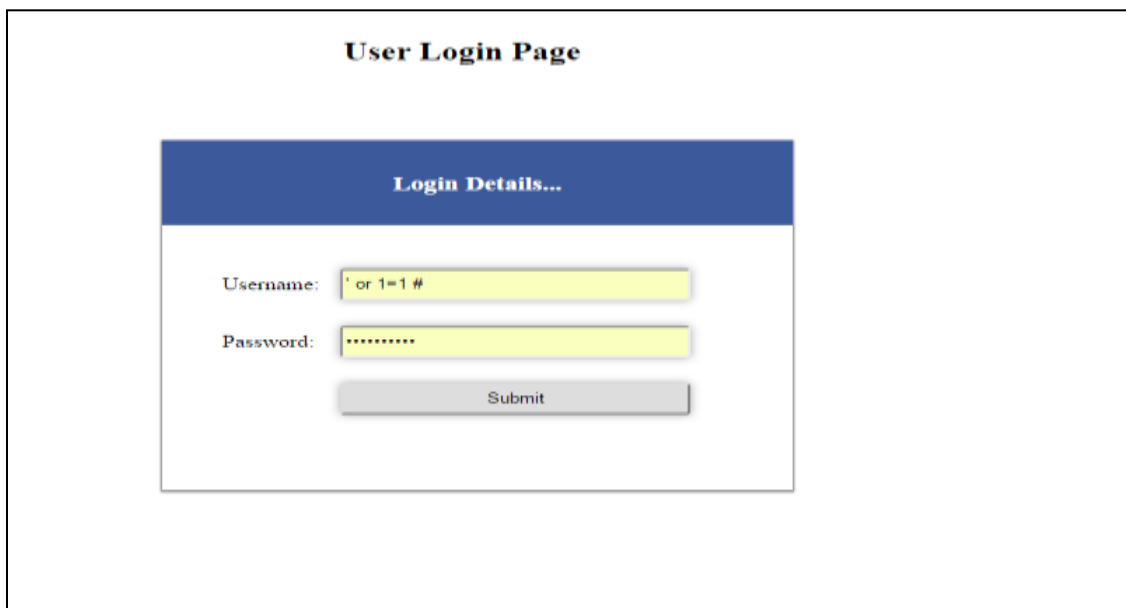


Figure5.4. User Login Page with Special Character Input

OUTPUT:

The output showing successful login is shown below in Figure5.5.

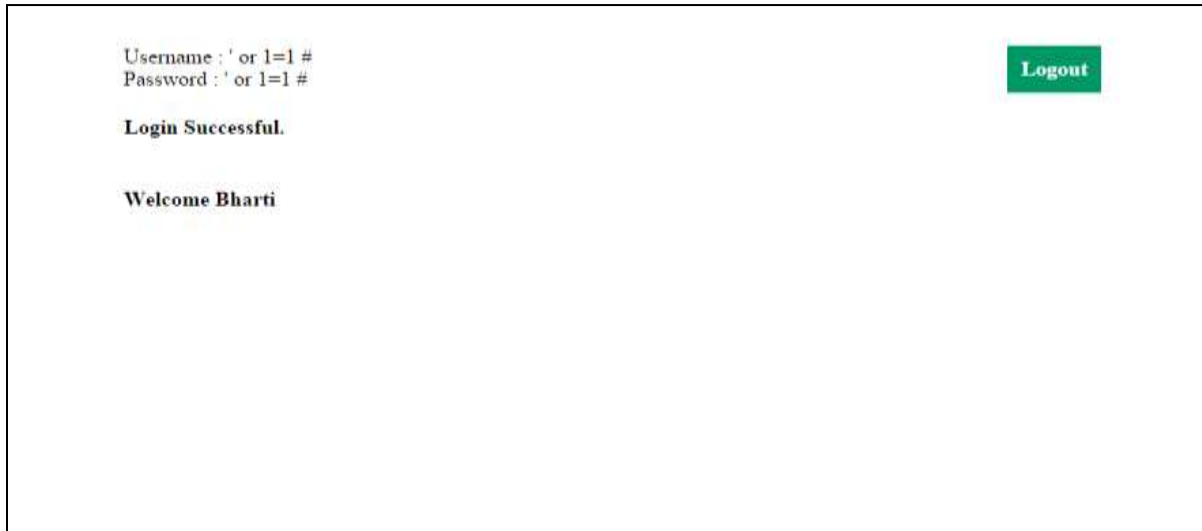


Figure5.5. Output showing successful login with special character

The above output shows successful login. From the output it can be observed that the code is vulnerable to SQL injection attack.

5.2.1.2 Preventing SQL Injection Vulnerability

The following snapshots shows the stepwise working of prevention of SQL injection attack using the proposed tool.

Step 1: Enter the Path of web application as shown below in Figure5.6

SQL FORM

1. Scan Folder / Path2. Scan For SQL INJECTION Files3. Static Analysis4. Dynamic Creation5. RUN / CHECK

Figure5.6. SQL form to enter web application path

Step 2: click on the option1.The output is shown below in Figure 5.7.

```
Path : C:\xampp\htdocs\login-website
Total Number of files : 6
Total Number of php files : 4
List of php files
C:\xampp\htdocs\login-website\db_connect.php
C:\xampp\htdocs\login-website/index.php
C:\xampp\htdocs\login-website/logout.php
C:\xampp\htdocs\login-website/verify.php
Step 1 completed. Please! move to next Step.
```

Figure5.7. Output generated after completion of step2

Step 3: click on the option2. The output is shown below in Figure 5.8.

```
Path : C:\xampp\htdocs\login-website
Total php files : 4
File Name : C:\xampp\htdocs\login-website\verify.php
$SQL = "select * from account where username ='$username' and password ='$password'";

Total number of select statement files : 1

Step 2 completed. Please! move to next Step.

Next

BACK
```

Figure5.8. Output generated after completion of step3

Step 4: click on the option3. The output is shown below in Figure 5.9.

```
Path : C:\xampp\htdocs\login-website
Total php files : 1
C:\xampp\htdocs\login-website\verify.php
Line Number : 6
$SQL = "select * from account where username ='$username' and password ='$password'";
Array ( [0] => select [1] => * [2] => from [3] => account [4] => where [5] => username [6] => = [7] => username [8] => and [9] => password [10] => = [11] => password )
Total Static token for the above line is 12

Step 3 completed. Please! move to next Step.

Next

BACK
```

Figure5.9. Output generated after completion of step4

Step 5: click on the option4. The output is shown below in Figure5.10.



Figure5.10. Output generated after completion of step5

Step 6: click on the option5 i.e Run option . The output is displayed.

The User Login Page with legitimate input is shown below using Figure 5.11.

INPUT:

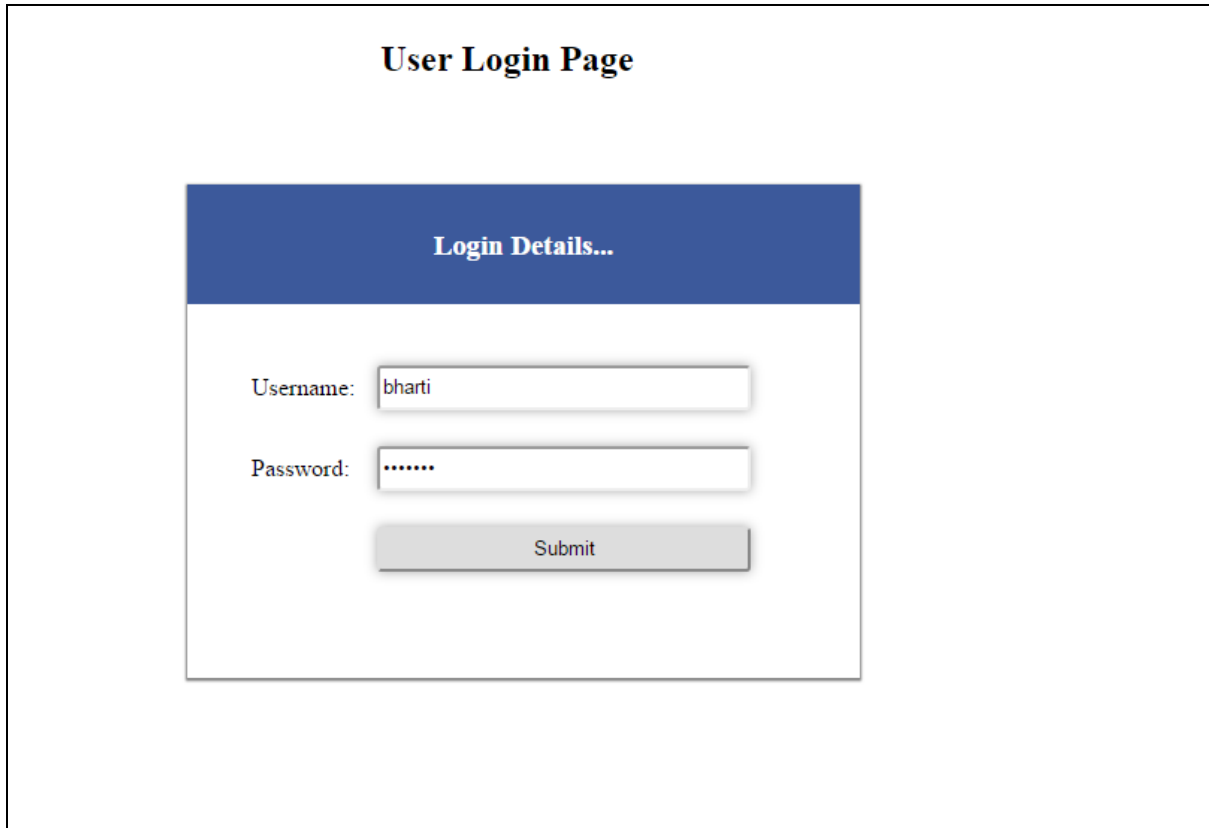


Figure5.11. User Login Page with legitimate input

OUTPUT:

Figure 5.12 shows successful login.

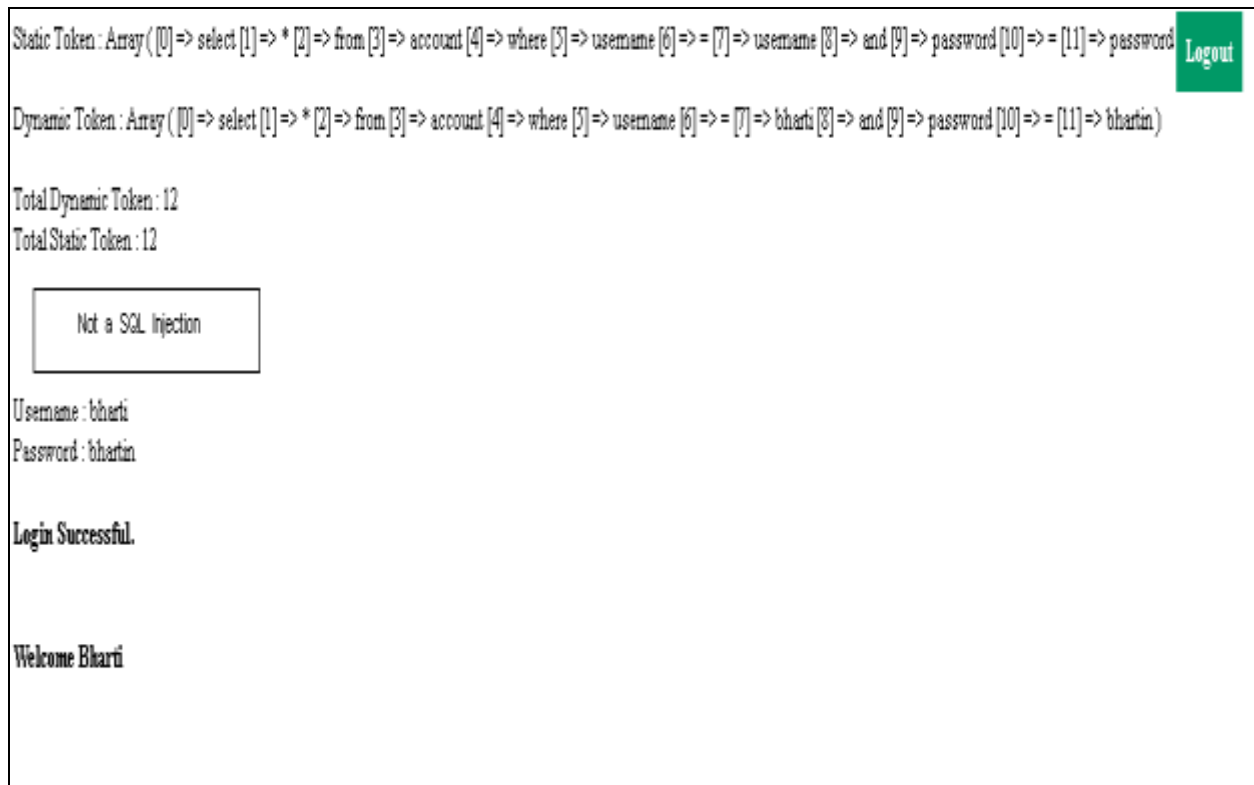


Figure5.12. Output showing successful login

From the above output it can be observed that this is not a SQL Injection.

The User Login Page with Tautology based non-legitimate input as shown below using Figure5.13.

INPUT:



Figure5.13. User Login Page with Tautology based non- legitimate input

OUTPUT:

The output showing SQL Injection attempted as shown below in Figure5.14.

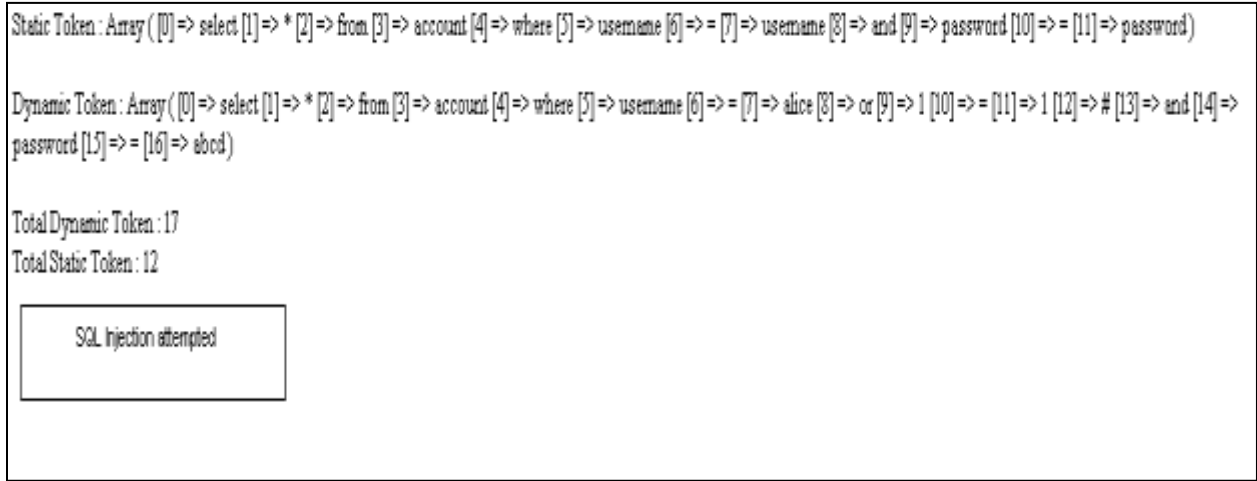
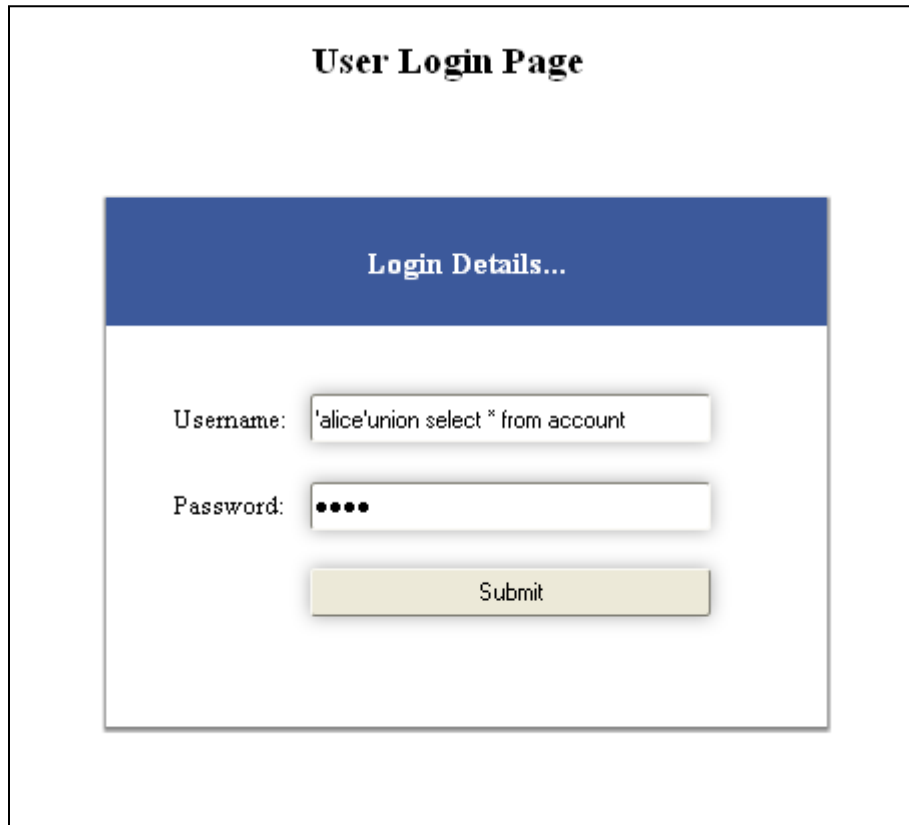


Figure5.14. Output showing SQL Injection attempted

The User Login Page with Union query based non-legitimate input as shown below using Figure5.15.

INPUT:



The image shows a web form titled "User Login Page". At the top, there is a blue header bar with the text "Login Details...". Below the header, there are two input fields. The first field is labeled "Username:" and contains the text "'alice'union select * from account". The second field is labeled "Password:" and contains five black dots. Below the password field is a yellow "Submit" button.

Figure5.15. User Login Page with union query based non- legitimate input

OUTPUT:

The output showing SQL Injection attempted as shown below in Figure5.16.

```
Static Token : Array ( [0] => select [1] => * [2] => from [3] => account [4] => where [5] => username [6] => = [7] => username [8] => and [9] => password [10] => = [11] => password )

Dynamic Token : Array ( [0] => select [1] => * [2] => from [3] => account [4] => where [5] => username [6] => = [7] => alice [8] => union [9] => select [10] => * [11] => from [12] => account [13]
=> and [14] => password [15] => = [16] => abcd )

Total Dynamic Token : 17
Total Static Token : 12

SQL injection attempted
```

Figure5.16. Output showing SQL Injection attempted

The User Login Page with Blind injection based non-legitimate input as shown below using Figure5.17.

INPUT:

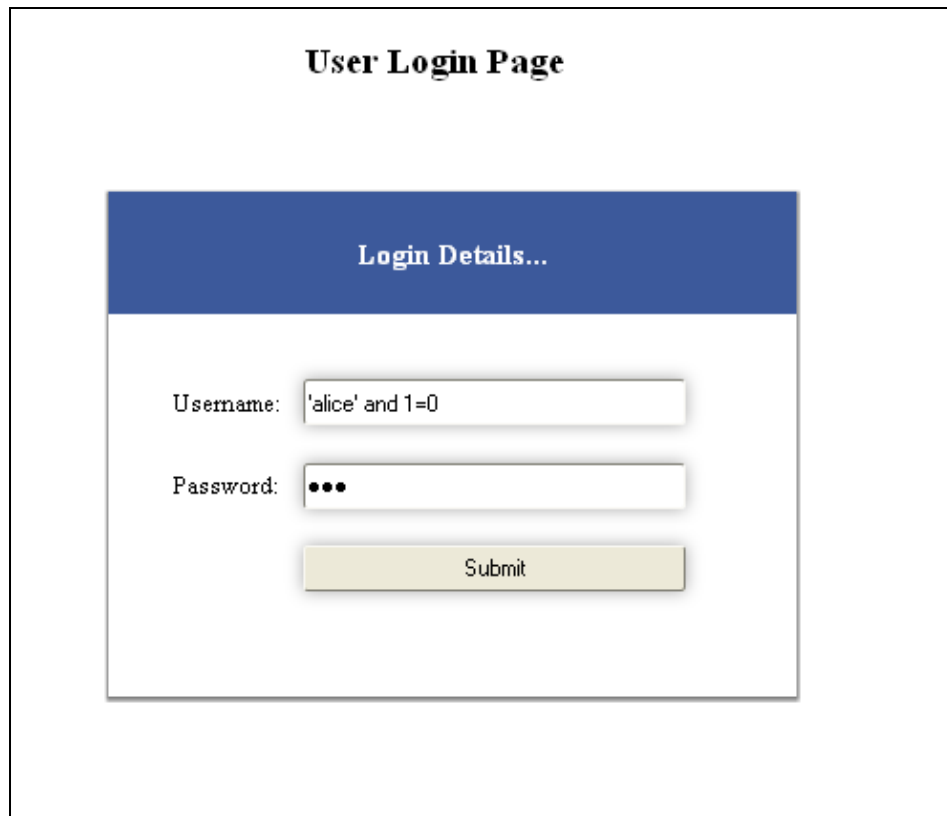


Figure5.17. User Login Page with blind injection based non-legitimate input

OUTPUT:

The output showing SQL Injection attempted as shown below in Figure5.18.

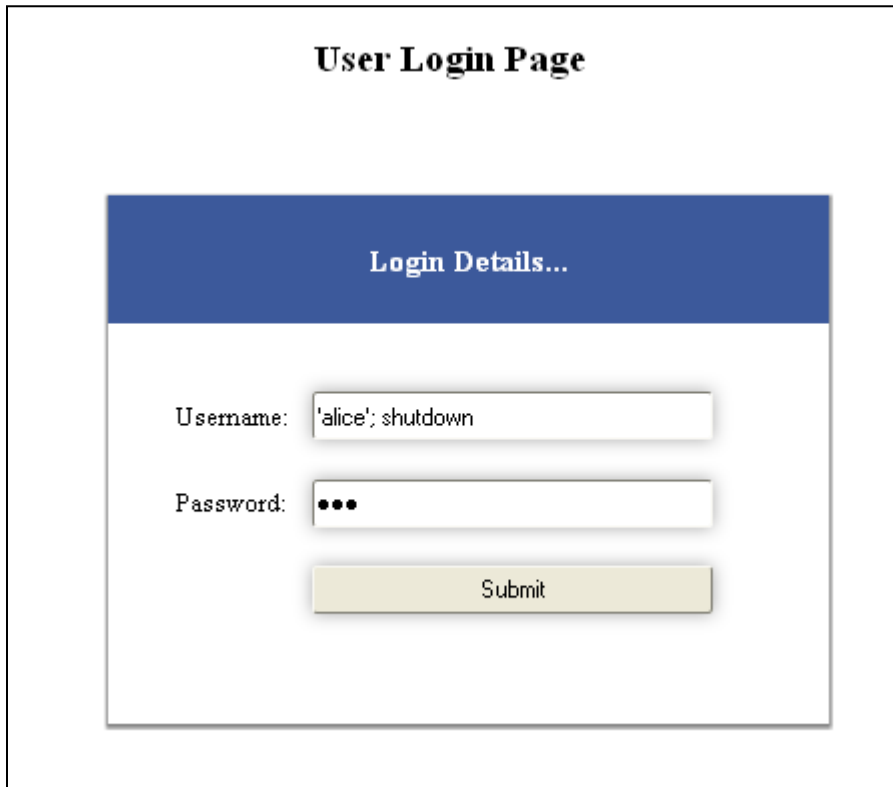
```

Static Token : Array ( [0] => select [1] => * [2] => from [3] => account [4] => where [5] => username [6] => = [7] => username [8] => and [9] => password [10] => = [11] => password )
Dynamic Token : Array ( [0] => select [1] => * [2] => from [3] => account [4] => where [5] => username [6] => = [7] => alice [8] => and [9] => 1=0 [10] => and [11] => password [12] => = [13] => xyz )
Total Dynamic Token : 14
Total Static Token : 12
SQL Injection attempted
  
```

Figure5.18. Output showing SQL Injection attempted

The User Login Page with Stored procedure based non-legitimate input as shown below using Figure5.19.

INPUT:



The image shows a web form titled "User Login Page". At the top, there is a blue header bar with the text "Login Details...". Below the header, there are two input fields. The first field is labeled "Username:" and contains the text "'alice'; shutdown". The second field is labeled "Password:" and contains three black dots. Below the password field is a yellow "Submit" button.

Figure5.19. User Login Page with stored procedure based non- legitimate input

OUTPUT:

The output showing SQL Injection attempted as shown below in Figure5.20.

```
Static Token : Array ( [0] => select [1] => * [2] => from [3] => account [4] => where [5] => username [6] => = [7] => username [8] => and [9] => password [10] => = [11] => password )

Dynamic Token : Array ( [0] => select [1] => * [2] => from [3] => account [4] => where [5] => username [6] => = [7] => alice [8] => , [9] => shutdown [10] => and [11] => password [12] => = [13]
=> xyz )

Total Dynamic Token : 14
Total Static Token : 12



SQL Injection attempted


```

Figure5.20. Output showing SQL Injection attempted

The User Login Page with Piggy-backed query based non-legitimate input as shown below using Figure5.21.

INPUT:

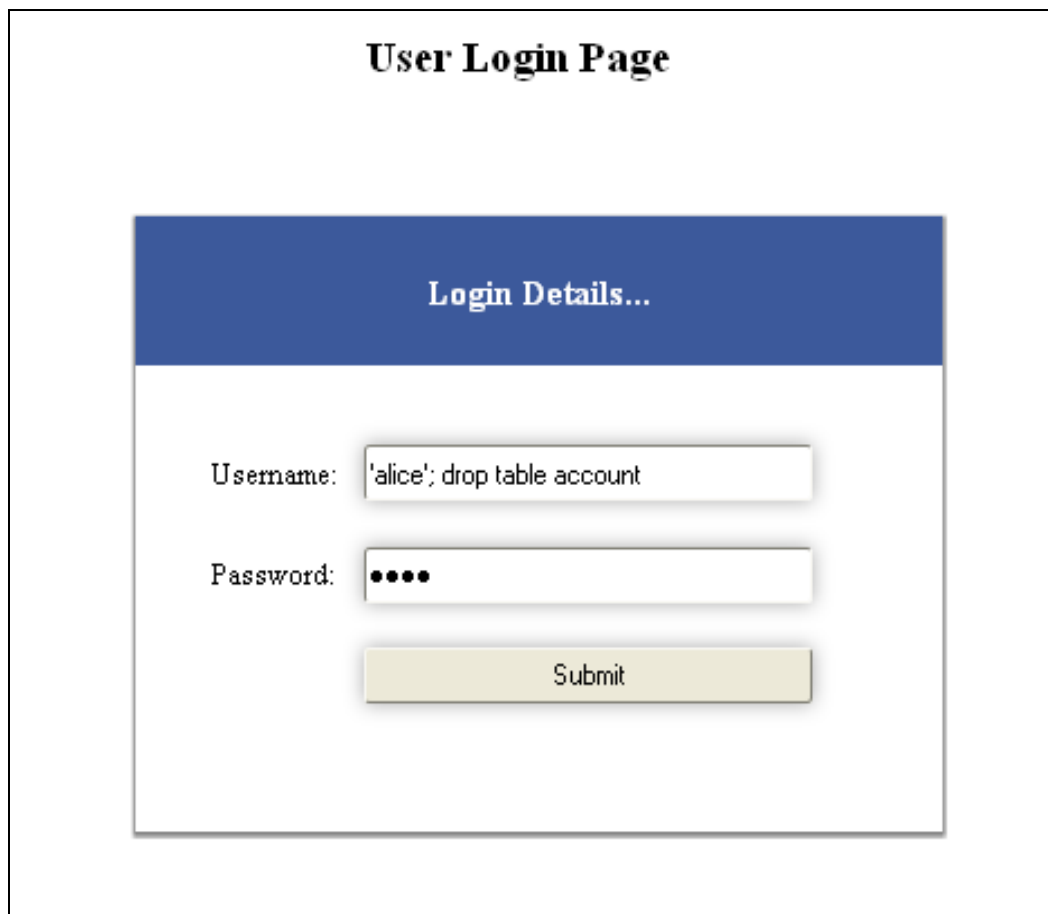


Figure5.21. User Login Page with Piggy-backed query based non- legitimate input

OUTPUT:

The output showing SQL Injection attempted as shown below in Figure5.22.


```
Static Token : Array ( [0] => select [1] => * [2] => from [3] => account [4] => where [5] => username [6] => = [7] => username [8] => and [9] => password [10] => = [11] => password )
Dynamic Token : Array ( [0] => select [1] => * [2] => from [3] => account [4] => where [5] => username [6] => = [7] => alice [8] => ; [9] => drop [10] => table [11] => account [12] => and [13] =>
password [14] => = [15] => abcd )
Total Dynamic Token : 16
Total Static Token : 12
SQL Injection attempted
```

Figure5.22. Output showing SQL Injection attempted

The above output shows the presence of SQLIA. An error report is generated and hence SQLI attack is prevented. Thus the non-legitimate user will not be allowed to access database.

5.2.2 XSS Vulnerability

It is a kind of injection in which hacker injects his own script code into a vulnerable website page. At the point when a victim visits this infected page in the web application just by browsing the web site, his browser downloads the hacker code and automatically executes it with accessing any file[19]. A hacker can send a malicious script to a non-suspecting client utilizing XSS. The end client browser does not have the possibility that the script ought not be trusted, and thus run the script. The malicious JavaScript appears as a legitimate component of web application by the victim's program. Hacker would be able to access data i.e. cookies, session id etc. after running the malicious script. [20,21]

5.2.2.1 Exploiting XSS Vulnerability

The user login page for inputting username and password is shown below using Figure 5.23.

INPUT:



The image shows a user login page titled "User Login Page". It features a central form with a blue header bar containing the text "Login Details...". Below the header, there are two input fields: "Username:" with the value "bharti" and "Password:" with a masked value of ".....". A "Submit" button is positioned below the password field.

Figure5.23. User Login Page with legitimate input

OUTPUT:

The output showing successful login as shown below in Figure5.24.

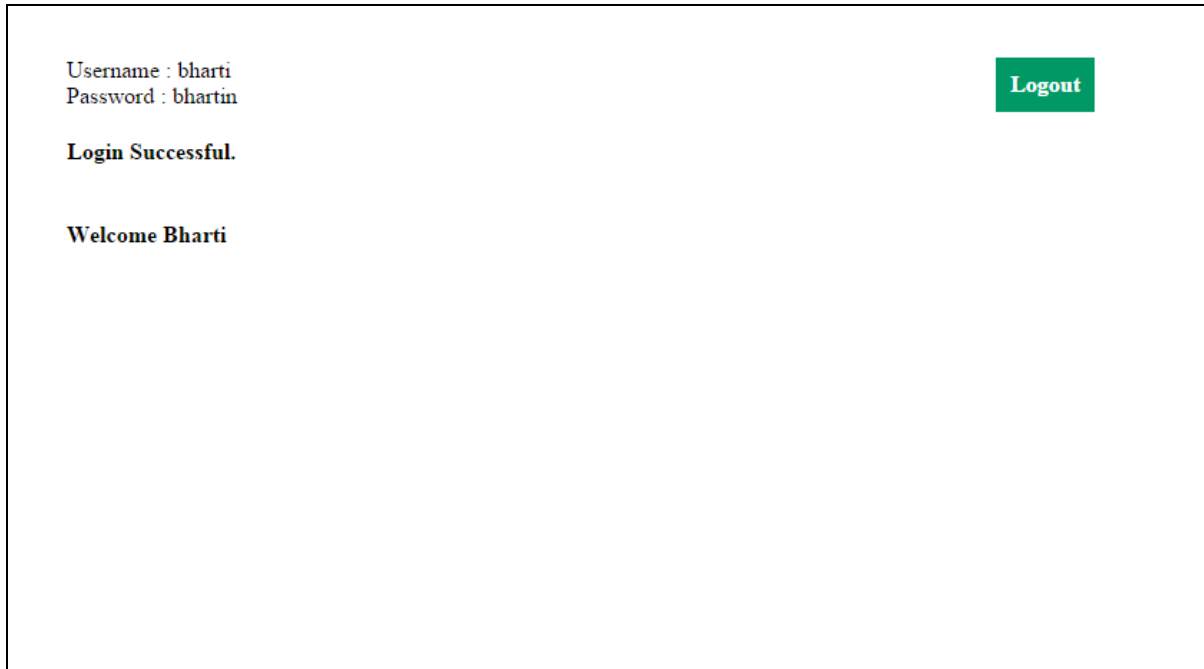


Figure5.24. Output showing successful login

The User Login Page with malicious input is shown below using Figure5.25.

INPUT:

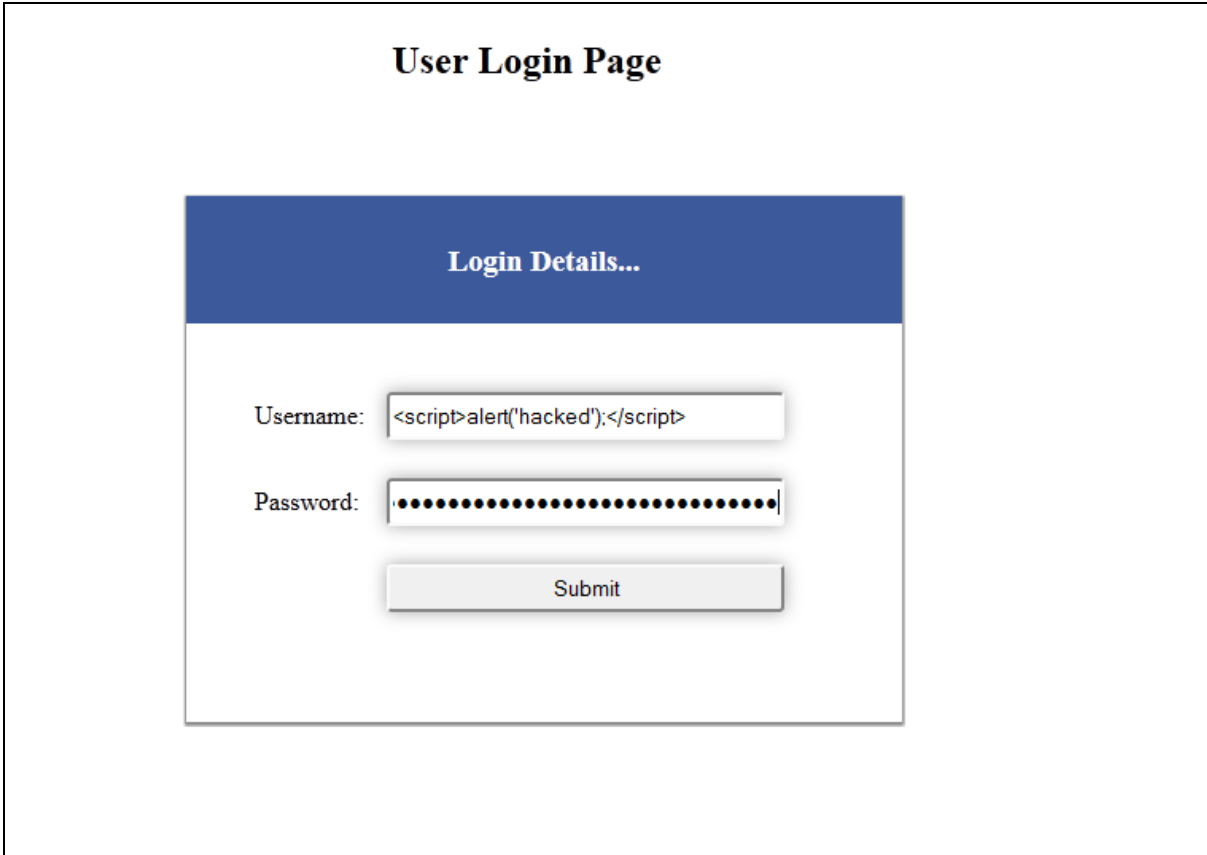


Figure5.25. User Login Page with malicious input

The output shown below in Figure 5.26 shows vulnerability to XSS attack.

OUTPUT:



Figure5.26.Output showing vulnerability to XSS attack

From the above output it can be observed that the code is vulnerable to XSS attack.

5.2.2.2 Preventing XSS Vulnerability

The following snapshots shows the stepwise working of prevention of XSS attack using the proposed tool.

Step 1: Enter the Path of web application as shown below in Figure 5.27.

XSS IMPLEMENTATION

C:\xampp\htdocs\login-website

1. Scan Folder / Path

2. Scan For XSS

3. Dynamic Creation

4. RUN / CHECK

BACK

Figure5.27. XSS form to enter web application path

Step 2: click on the option1.The output is shown below in Figure 5.28.

```
Path : C:\xampp\htdocs\login-website
Total Number of files : 6
Total Number of php files : 4
List of php files
C:\xampp\htdocs\login-website/db_connect.php
C:\xampp\htdocs\login-website/index.php
C:\xampp\htdocs\login-website/logout.php
C:\xampp\htdocs\login-website/verify.php
```

Step 1 completed. Please! move to next Step.

Next

BACK

Figure5.28. Output generated after completion of step2

Step 3: click on the option2. The output is shown below in Figure5.29.



Figure5.29. Output generated after completion of step3

Step 4: click on the option3. The output is shown below in Figure5.30.

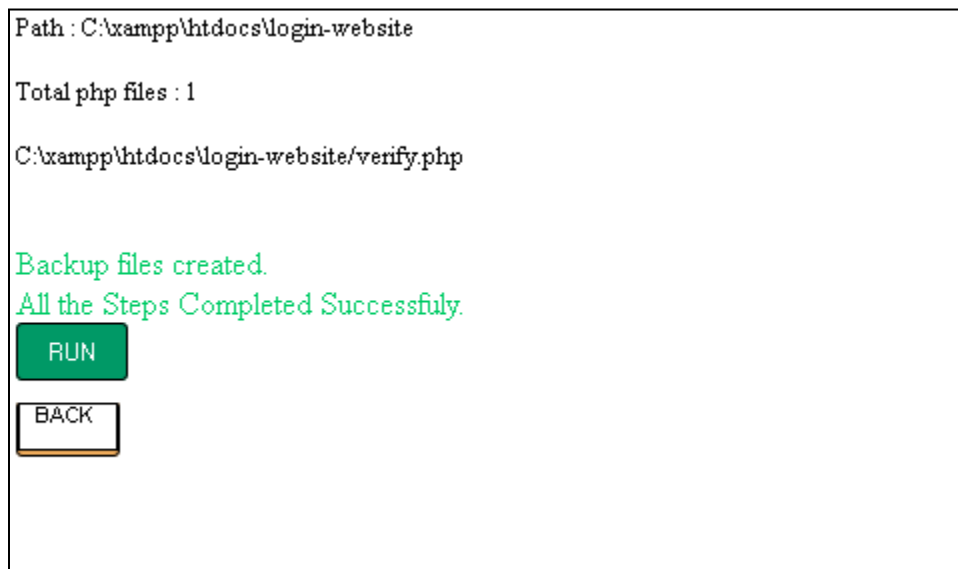


Figure5.30. Output generated after completion of step4

Step 5: click on the option4 i.e Run option . The output is displayed.

Figure 5.31 shows User Login Page with legitimate input.

INPUT:



The image shows a screenshot of a web application's login page. At the top, the title "User Login Page" is centered. Below it is a blue header bar with the text "Login Details...". The main content area contains two input fields: "Username:" with the value "bharti" and "Password:" with a masked password represented by seven dots. Below the password field is a "Submit" button.

Figure5.31. User Login Page with legitimate input

OUTPUT:

Figure 5.32 shows successful login.

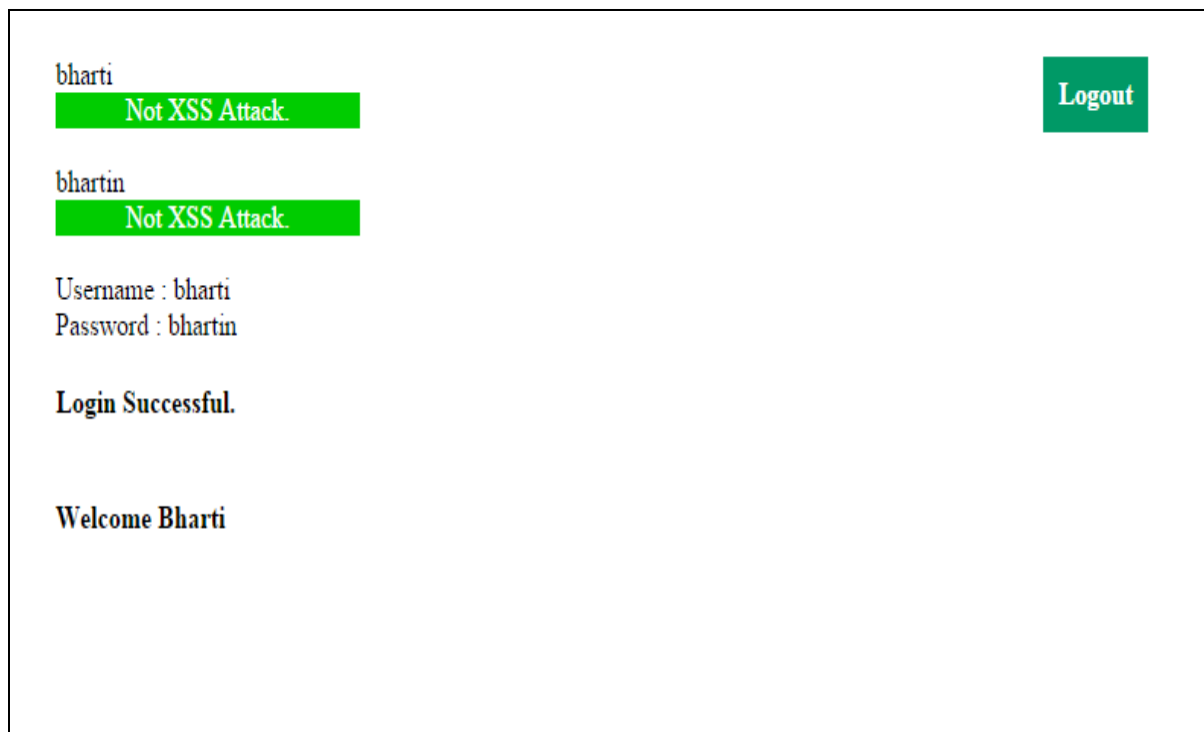


Figure5.32. Output showing successful login

The above output shows successful login.

Figure 5.33 shows User Login Page with malicious script tag input.

INPUT:

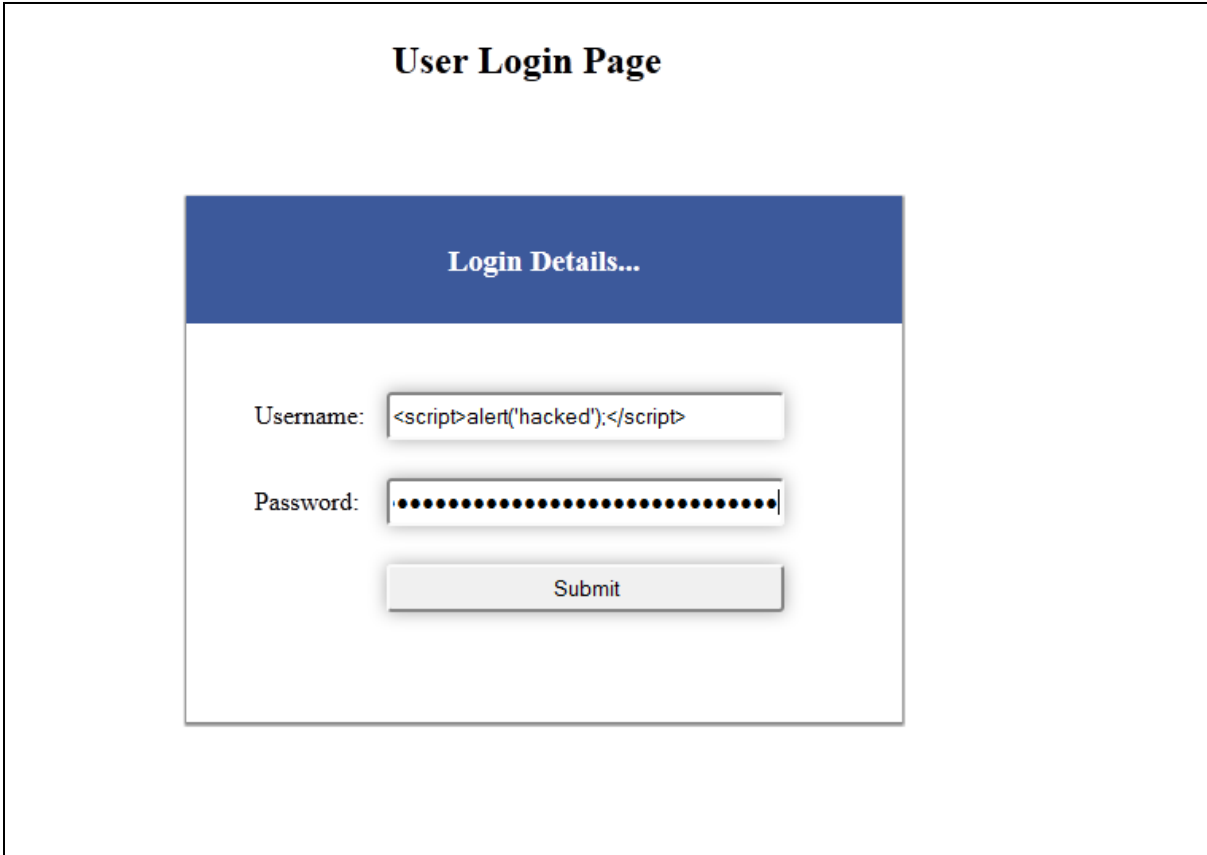


Figure5.33. User Login Page with malicious script tag input

OUTPUT:

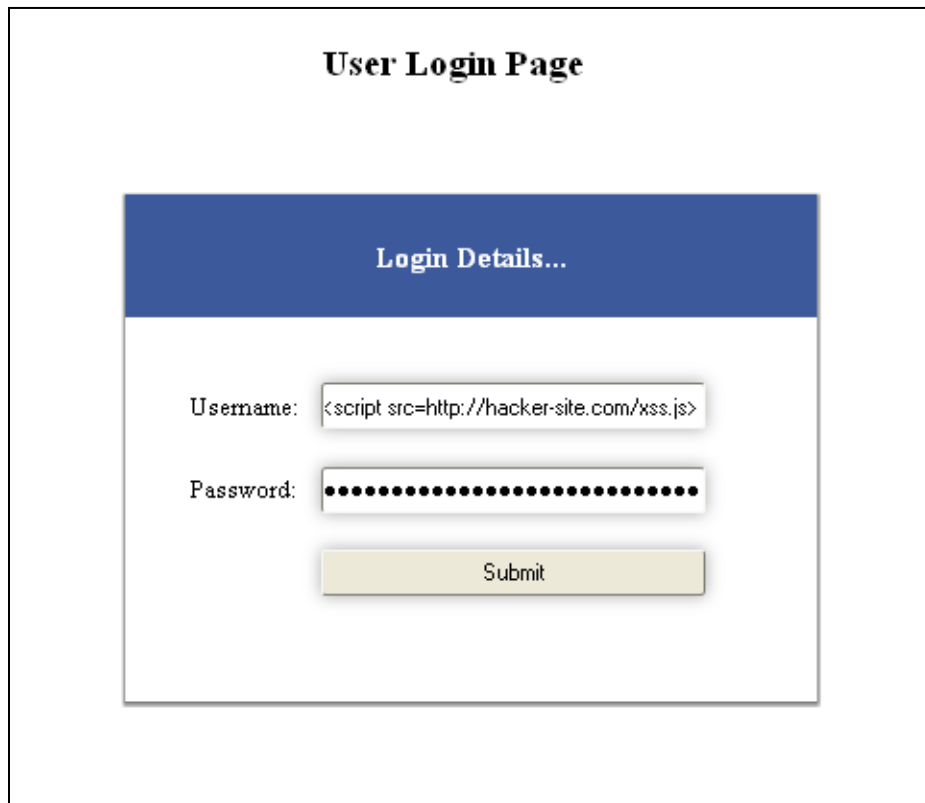
The output showing XSS attack attempted as shown below in Figure5.34.



Figure5.34. Output showing XSS attack attempted

Figure 5.35 shows User Login Page with malicious source tag input.

INPUT:



The image shows a web form titled "User Login Page". At the top, there is a blue header bar with the text "Login Details...". Below the header, there are two input fields. The first field is labeled "Username:" and contains the text "<script src=http://hacker-site.com/xss.js>". The second field is labeled "Password:" and contains a series of black dots. Below the password field is a "Submit" button.

Figure5.35. User Login Page with malicious source tag input

OUTPUT:

The output showing XSS attack attempted as shown below in Figure5.36.

OUTPUT:

The output showing XSS attack attempted as shown below in Figure5.38.



Figure5.38. Output showing XSS attack attempted

Figure 5.39 shows User Login Page with malicious image tag input.

INPUT:



Figure5.39. User Login Page with malicious image tag input

OUTPUT:

The output showing XSS attack attempted as shown below in Figure5.40.



Figure5.40. Output showing XSS attack attempted

Figure 5.41 shows User Login Page with malicious iframe tag input.

INPUT:



Figure5.41. User Login Page with malicious iframe tag input

OUTPUT:

The output showing XSS attack attempted as shown below in Figure5.42.



Figure5.42. Output showing XSS attack attempted

Figure 5.43 shows User Login Page with malicious div tag input.

INPUT:

User Login Page

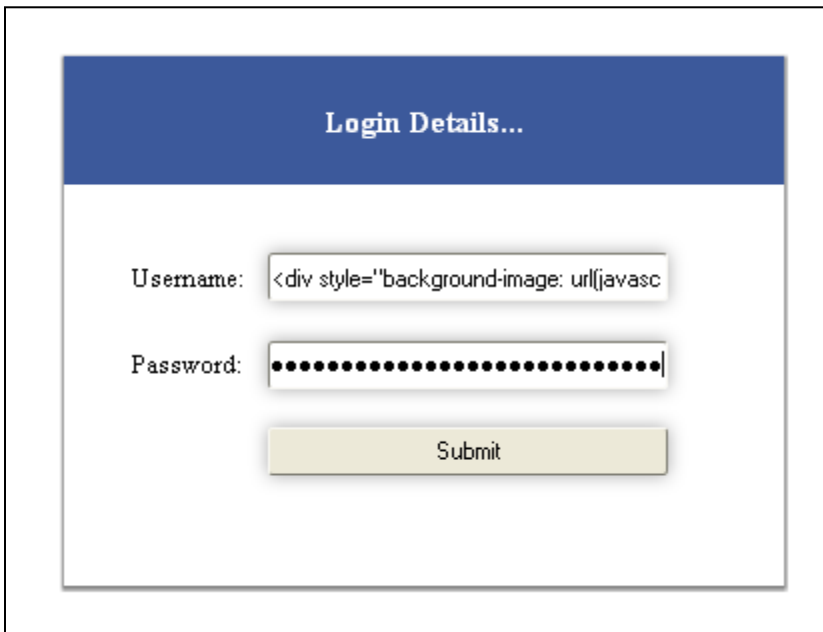


Figure5.43. User Login Page with malicious div tag input

OUTPUT:

The output showing XSS attack attempted as shown below in Figure5.44.



Figure5.44. Output showing XSS attack attempted

Figure 5.45 shows User Login Page with malicious embed tag input.

INPUT:

User Login Page

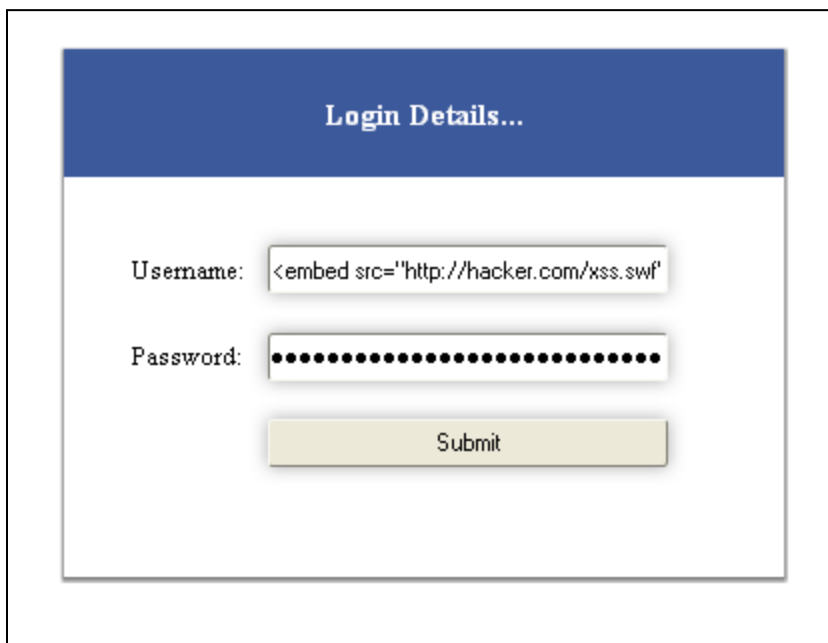


Figure5.45. User Login Page with malicious embed tag input

OUTPUT:

The output showing XSS attack attempted as shown below in Figure5.46.



Figure5.46. Output showing XSS attack attempted

The above output shows the presence of XSS attack. An error report is generated and hence XSS attack is prevented. Thus the non-legitimate user will not be allowed to access the credentials.

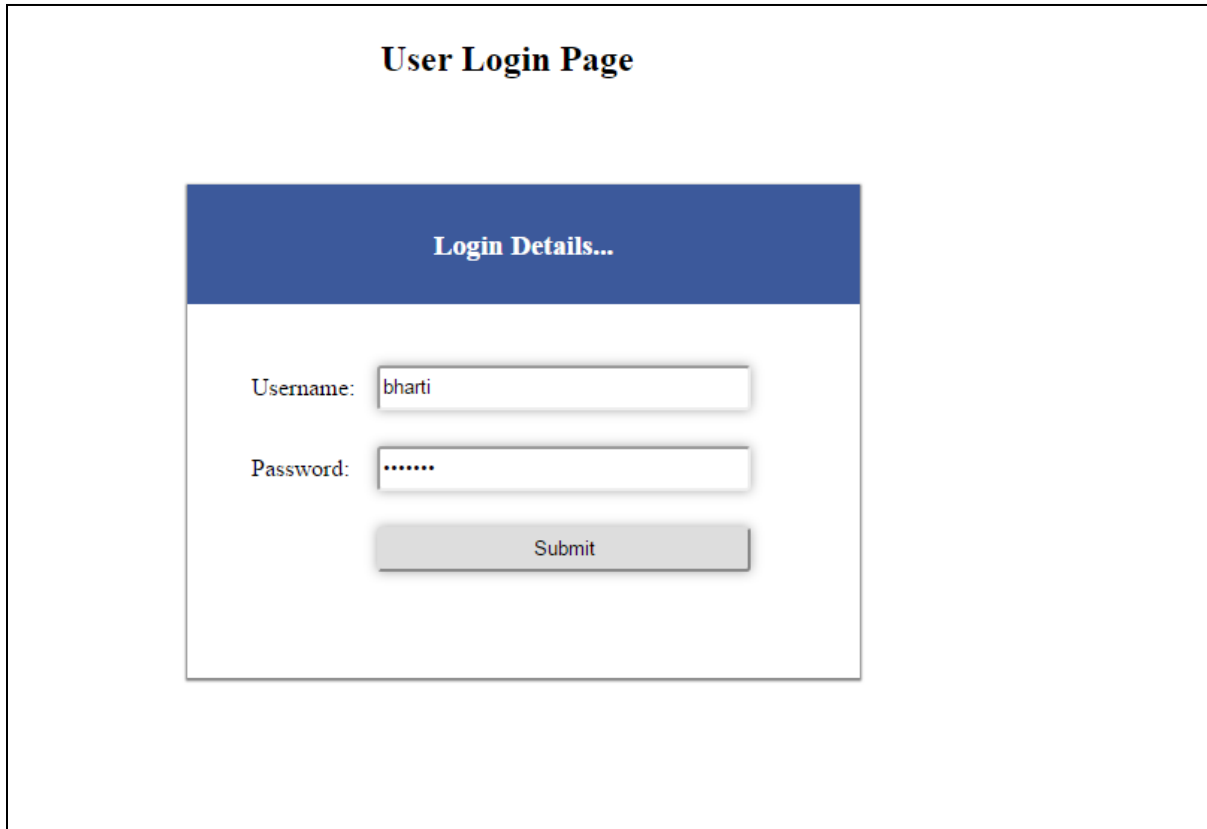
5.2.3 CSRF Vulnerability

CSRF attack arises when a non trusted website causes a client's web browser to permit a malicious activity on a trusted website. This is due to the fake HTTP request as it exploits the currently running client's session of the web browser. A CSRF web attack requires inclusion of three things. A target client, a trustable website, and a non trustable web site. The target client is currently holding an active session with a trustable site and in the meanwhile, the client visits a malicious or non trusted website. The non trustable or malicious web site injects a HTTP request for the trustable web site into the target client's session which compromises its integrity. These vulnerabilities permit a hacker to exchange money out from client's account, to collect client's email id, disregard client privacy etc.

5.2.3.1 Exploiting CSRF Vulnerability

User login page for inputting username and password is shown below using Figure 5.47.

INPUT:



The image shows a user login page titled "User Login Page". It features a blue header bar with the text "Login Details...". Below the header, there are two input fields: "Username:" with the value "bharti" and "Password:" with a masked password represented by seven dots. A "Submit" button is located below the password field.

Figure5.47. User Login Page with legitimate input

OUTPUT:

Figure 5.48 shows successful login.

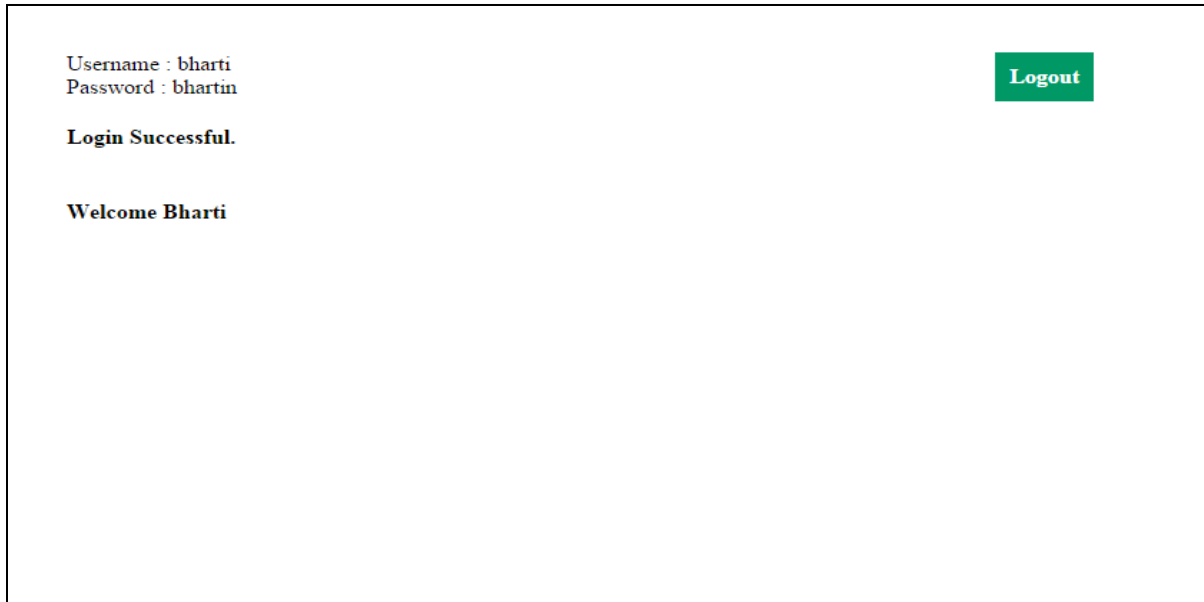


Figure5.48. Output showing successful login

5.2.3.2 Preventing CSRF Vulnerability

The following snapshots shows the stepwise working of prevention of CSRF attack using the proposed tool.

Step 1: Enter the Path of web application as shown below in Figure 5.49.



Figure5.49. CSRF form to enter web application path

Step 2: click on the option1.The output is shown below in Figure 5.50.



Figure5.50. Output generated after completion of step2

Step 3: click on the option2. The output is shown below in Figure 5.51.



Figure5.51. Output generated after completion of step3

Step 4: click on the option3. The output is shown below in Figure 5.52.

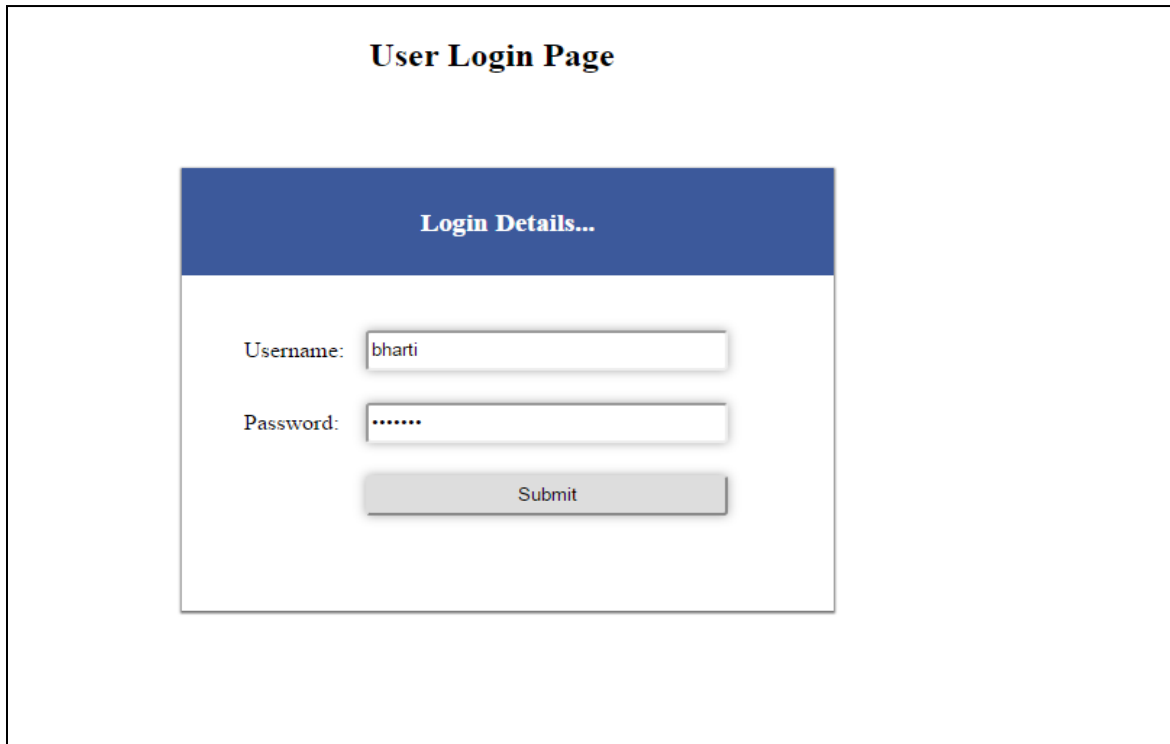


Figure5.52. Output generated after completion of step4

Step 5: click on the option4 i.e Run option . The output is shown below.

Figure 5.53 shows User Login Page with legitimate input.

INPUT:



The image shows a web page titled "User Login Page". At the top, there is a blue header bar with the text "Login Details...". Below the header, there are two input fields. The first is labeled "Username:" and contains the text "bharti". The second is labeled "Password:" and contains seven dots. Below the password field is a grey "Submit" button.

Figure5.53. User Login Page with legitimate input

OUTPUT:

Figure 5.54 shows successful login.



Figure5.54. Output showing successful login

Figure 5.55 shows User Login Page with malicious input.

INPUT:

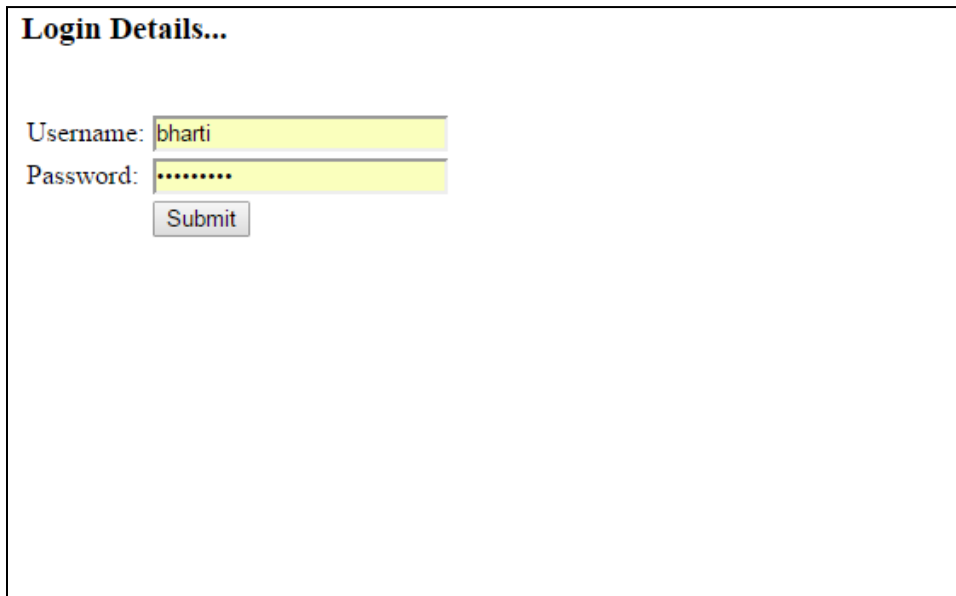


Figure5.55. User Login Page with malicious input

OUTPUT:

The output showing CSRF attack attempted as shown below in Figure5.56.

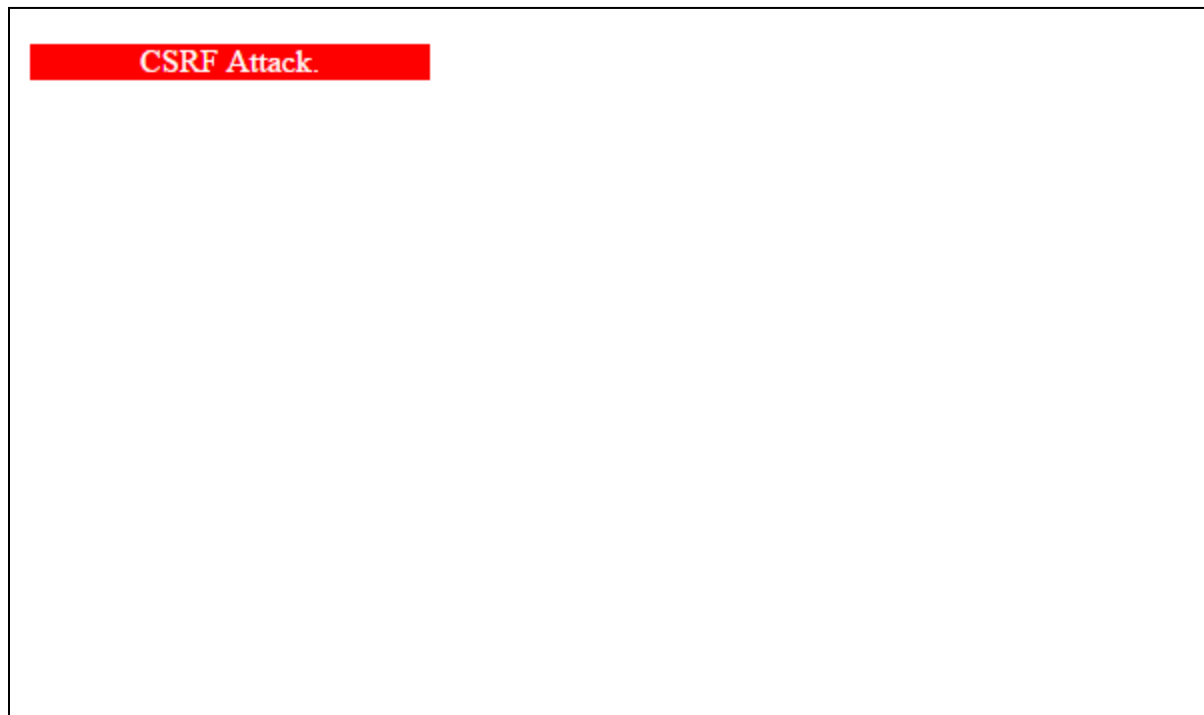


Figure5.56. Output showing CSRF attack attempted

From the above output it can be observed that there is an attempt of CSRF attack. An error report is generated and hence CSRF attack is prevented. Thus the non-legitimate user will not be allowed to access the credentials.

5.3 EXPERIMENTAL AND COMPARATIVE ANALYSIS

In this section, test-bed is used to analyze the efficiency of the methodology which has been proposed. This test bed is a set of web applications which are vulnerable to SQLI, XSS and CSRF attacks along with test inputs that represent malicious and legitimate access to the web application.

5.3.1 Test Input Generation

As per the proposed methodology, every application consists of two sets of test inputs. The first testing set is having the HTTP request statements containing valid input. They are required during static mode so as to construct model[76]. It is the developer's responsibility to consider all kind of test inputs so that they can cover every hotspot in the application. The second testing set consists of malicious SQLI, XSS and CSRF HTTP request statements.

5.3.2 Web Application Testing And Results

To analyze the efficacy of the methodology which has been proposed, results are calculated on different set of PHP applications having multiple complexities. The web applications selected are exposed to SQLI attack. Similarly, the web applications selected are exposed to XSS attack and finally, the web applications selected are exposed to CSRF attack. Experimental results for SQLIA is mentioned below in Table5.1. Similarly, experimental results for XSS attack is mentioned below in Table5.2. The experimental results for CSRF attack is mentioned below in Table5.3. The instrumentation overhead is defined as the %age of code which is appended to the original code. Query execution overhead is computed as the percentage increase in the time needed for running queries of modified web application to the time needed for running queries in original web application. False positive is to detect an attack even if it does not exist.

Table5.1 Experimental analysis for SQLIA

Web Application	Lines of code (K)	Hotspots Instrumented	Instrumentation overhead(%)	Query execution overhead(%)	Prevention (%)	False Positive
College portal	1.2	25	8	1.05	100	0
ToyRental	7.6	40	10	2.10	100	0
BuyMilkOnline	4.5	20	7	1.80	100	0

Table5.2 Experimental analysis for Cross Site Script attack

Web Application	Lines of code (K)	Hotspots Instrumented	Instrumentation overhead(%)	Query execution overhead(%)	Prevention (%)	False Positive
College portal	1.2	17	6	1.01	100	0
ToyRental	7.6	32	7	1.91	100	0
BuyMilkOnline	4.5	13	5	1.66	100	0

Table5.3 Experimental results for Cross Site Request Forgery attack

Web Application	Lines of code (K)	Hotspots Instrumented	Instrumentation overhead(%)	Query execution overhead(%)	Prevention (%)	False Positive
College portal	1.2	30	8	1.30	100	0
ToyRental	7.6	45	10	2.21	100	0
BuyMilkOnline	4.5	35	6	1.80	100	0

The efficiency of the solution which is proposed is tested by noticing the total attacks which are prevented to the total attacks which are performed. From the experimental analysis mentioned above, it can be seen that the security system is 100% effective to prevent the most commonly found serious and dangerous web attacks namely SQLI ,XSS as well as CSRF attack with very little overhead and no false alarms.

5.3.3 Comparative Analysis

The comparative analysis of different techniques/approaches proposed by different researchers is shown below in Table5.4.

Table5.4 Comparative analysis of different techniques/approaches

S.No.	Technique/Approach	Attack	Advantages	Disadvantages	Prevention
1.	Runtime monitors for tautology[51]	SQLI	Simple to implement and less complex.	For java applications only, detects tautology only, incomplete implementation	Yes
2.	SQLIMW[50]	SQLI	More flexible and scalable, less computation time	Works for sign-in applications only	Yes
3.	Blueprint[42]	XSS	Robust prevention approach	Possibility of false positive	Yes
4.	Client based proxy [74]	CSRF	Easy to monitor attack	Chances to lose sensitive information	No
5.	Randomizing the instruction set[69]	CSRF	Easy to implement	Possibility of false positive	Yes
6.	Dynamic cookie rewriting[19]	XSS	Effective technique	Not tested with HTTP connection	Yes
7.	CANDID[56]	SQLI	Less complex	Performance issues,less efficient	Yes
8.	Integrated approach for SQLI and	SQLI,	Less complex	Doesn't work for zero day	Yes

	reflected XSS[49]	Reflected XSS		exploits, false positive	
9.	SQLDOM[63]	SQLI	Efficient in solving compiler errors	Developer learning is required, increased runtime cost	No
10.	Browser protection[27]	CSRF	Easy to implement	Plugins may get crushed	Yes
11.	Proposed Hybrid Security System[76]	SQLI,XSS, CSRF	Easy to implement,no false positive, less runtime overhead	Works for PHP applications and for known attacks	Yes

The comparative analysis shows that the proposed Hybrid Security System prevents three attacks whereas other techniques prevent either one or two attack. The proposed hybrid security system which is a combination of three attacks prevents SQLI, XSS and CSRF attack with very little overhead and no false positives whereas other techniques shows false positives.

5.4 CONCLUSION

This chapter discussed the implementation of hybrid security system. Later on experimental analysis and comparative analysis for different web attacks is performed. To analyze the efficiency of proposed method, results are evaluated on different web based applications. The experimental analysis shows that the hybrid security system is 100% effective in preventing the web attacks namely SQLI , XSS as well as CSRF attack with very little overhead and no false alarms. The detailed conclusion of the work proposed and the possibilities of future research work is discussed in the next chapter.

CHAPTER VI

CONCLUSIONS AND FUTURE SCOPE

6.1 CONCLUSIONS

This chapter presents the major achievements of the research work and provides an outlook to further research in this area. The research work has given a security system for web attacks.

Web applications are often vulnerable to perform attacks, which further give hackers to easy access to the database. In light of the expanded number of assaults exploiting, many endeavors have been made to discover solution for the issue. The best arrangement is to create the programs in a safe way. Many archives have been distributed in regard to secure advancement of web based applications although very little has managed. Web engineers are not yet security mindful, and the issues keep on appearing. Accordingly, security administrators are continuously searching for different measures that can be taken against this issue. Developers are not yet security aware, and the issues continue to appear. Thus security experts constantly looking for some other countermeasures which can be considered against the problem.

An in-depth literature work was carried out and the critical analysis of the same raised the following objectives:

- To propose framework for Security system
- To propose prevention techniques for the most commonly found serious and dangerous web attacks such are Cross Site Script i.e. XSS, SQL Injection i.e. SQLI, Cross-Site Request Forgery i.e. CSRF on web applications.
- To perform validation and to generate results and error report.

- To perform evaluation of security system with a set of web applications of different complexities.
- To perform comparative analysis of security system

In the light of the objectives identified and in order to counter the increased number of attacks taking advantage of the confidential access of information, a security system for the most commonly found serious and dangerous web based attacks is proposed. It is a hybrid system which is developed in PHP. This hybrid security system prevents the most commonly found serious and dangerous web based attacks which are Cross Site Script i.e. XSS, SQL Injection i.e. SQLI, Cross-Site Request Forgery i.e. CSRF in a more efficient way by reducing the drawbacks of the existing techniques given by different researchers and thereby to improve performance. The proposed hybrid system works in different phases which leads to easy design and implementation.

To analyze the efficacy of the methodology which has been proposed, results are calculated on different set of PHP applications having multiple complexities. The efficiency of the solution which is proposed is tested by noticing the total attacks which are prevented to the total attacks which are performed. From the experimental analysis mentioned above, it can be seen that the security system is 100% effective to prevent the most commonly found serious and dangerous web attacks namely SQLI ,XSS as well as CSRF attack with very little overhead and no false alarms.

6.2 BENEFITS OF PROPOSED DESIGN

The significant achievements of the proposed design are listed below:

- A security system which is 100% effective to prevent the most commonly found serious and dangerous web attacks namely SQL injection ,XSS and CSRF.
- Simple framework
- Complete implementation
- Less overhead
- No false alarms

6.3 FUTURE SCOPE

The work contained in the thesis made an attempt to answer the questions which came forward as a result of literature survey. The work can be extended with the further research in the near future as mentioned below-

- The proposed Hybrid Security System counter web based attacks which are Cross Site Script i.e. XSS, SQL Injection i.e. SQLI, Cross-Site Request Forgery i.e. CSRF. This hybrid security system is specifically for PHP applications. The implementation can be done using different scripting languages other than PHP depending upon the requirement.
- The proposed Hybrid Security System works for known attacks. It can be extended for zero day exploits.
- The Hybrid Security System can be expanded for more web vulnerabilities and attacks which are listed by OWASP.

REFERENCES

- [1] [www.open.edu / openlearn / science-maths-technology /computing-and-ict /introduction-information-security/content-section](http://www.open.edu/openlearn/science-maths-technology/computing-and-ict/introduction-information-security/content-section)
- [2] <http://www.slideshare.net/cchamnap/introduction-to-web-architecture>
- [3] <https://www.techopedia.com/definition/24649/three-tier-architecture>
- [4] J.G. Kim, "Injection Attack Detection using the Removal of SQL Query Attribute Values," in *11th IEEE conference on Information Science and Applications(ICISA)*, April 2011.
- [5] The Open Web Application Security Project, "OWASP TOP 10 Project", <http://www.owasp.org/>
- [6] [http://www.sans.org/reading-room/whitepaper/application/web based attacks](http://www.sans.org/reading-room/whitepaper/application/web-based-attacks)
- [7] <http://www.testingexcellence.com/static-analysis-vs-dynamic-analysis-software-testing/>
- [8] M.Christodorescu, S. Jha, "Static analysis of executables to detect malicious patterns," in *12th USENIX Security Symposium(Security'03)*, ACM, August 2003, pp.169-186.
- [9] F.Bellard, "Qemu , a Fast and Portable Dynamic Translator," *In USENIX Annual Technical Conference*, ACM, April 2005.
- [10] G. Kniesel, "Type-Safe Delegation for Run-Time Component Adaptation," in *European Conference on Object-Oriented Programming*, Springer, November 1999, pp. 351–366.
- [11] S.O. Al-Mamory, H. Zhang, "Intrusion Detection Alarms Reduction Using Root Cause Analysis And Clustering," in *Computer Communications*, Elsevier, Vol. 32, No.2, February

2009, pp. 419-430.

- [12] B . Nagpal, N. Chauhan, N. Singh, P .Sharma, "Preventive Measures For Securing Web Applications Using Broken Authentication And Session Management Attacks: A Study, " in *International Conference on Advances in Computer Engineering and Applications (ICACEA-2014)*,Feb2014.
- [13] A. Shrestha, P.S. Maharjan , S. Paudel , "Identification and Illustration of Insecure Direct Object References and their Countermeasures, " *International Journal of Computer Applications (IJCA)*, Vol. 114, No.18, , March 2015.
- [14] B. Eshete,A. Villafiorita,K Weldemariam, " Early Detection of Security Misconfiguration Vulnerabilities in Web Applications , " in *6th IEEE International Conference on Availability, Reliability and Security (ARES)*, August 2011.
- [15] X.Shu , D.Yao, E. Bertino," Privacy-Preserving Detection of Sensitive Data Exposure," *IEEE Transactions on Information Forensics and Security*,Vol. 10, No. 5, 2015, pp 1092-1103.
- [16] H.T.Le, C.D. Nguyen, L.Briand, B.Hourte , " Automated Inference of Access Control Policies for Web Applications," in *20th ACM Symposium on Access Control Models and Technologies*, June 2015, pp. 27-37.
- [17] I. Chowdhury, M. Zulkernine , " Using Complexity, Coupling, and Cohesion metrics as Early Indicators of Vulnerabilities , " *Journal of Systems Architecture*, Elsevier, Vol. 57, No. 3, March 2011.
- [18] J. Wang , H. Wu , " URFDS : Systematic discovery of Unvalidated Redirects and Forwards in web applications , " in *IEEE Conference on Communications Network and Security (CNS)*, September 2015.
- [19] R. Putthacharoen , P. Bunyatneparat , " Protecting Cookies from Cross Site Script Attacks using Dynamic Cookies Rewriting Technique , " in *13th IEEE International Conference on Advanced Communication Technology (ICACT)*, February 2011.

- [20] Q. Z. H. Chen , J. Sun , “ An Execution flow Based Method for Detecting Cross Site Scripting Attacks, ” in *2nd IEEE International Conference on Software Engineering and Data Mining (SEDM)*, June 2010, pp160-165.
- [21] B.Nagpal,N.Chauhan ,N. Singh,"Approaches to Detect and Prevent Cross-Site Scripting Attacks on Websites : A Survey, ” *I-Manager’s Journal on Information Technology (JIT)*,ISSN :2277-5110,Vol.2,No.4, 2013,pp 36-43.
- [22] D. Aucsmith ,” Creating and maintaining software that resists malicious attack,” *Distinguished Lecture Series*,Atlanta GA,September 2004.
http://www.gtisc.gatech.edu/aucsmith_bio.htm
- [23] G. Jiao , C. M. XU, J. Msohua ,“ SQLIMW : a new mechanism against SQL Injection , “ in *IEEE International Conference on Computer Science and Service System*, August2012.
- [24] B. Nagpal ,N.Chauhan,N. Singh," A Survey on the Detection of SQL Injection Attacks and Their Countermeasures,”*Journal of Information Processing Systems(JIPS)*,Korea, ISSN: 1976-913X, Vol.13, No.4, 2017, pp 689-702.
- [25] B. Nagpal , N. Singh ,N. Chauhan , A. Panesar ,“ Tool based Implementation of SQL Injection for Penetration Testing, ” in *IEEE International Conference on Computing, Communication and Automation(ICCCA-2015)*,2015.
- [26] A. Barth,C.Jackson,J.C.Mitchell,“ Robust Defenses for Cross-Site Request Forgery ,” in *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS’08)*, October 2008,pp 75-88.
- [27] W. Maes, T. Heyman, L. Desmet, W. Joosen, “ Browser protection against cross-site request forgery,” in *proceedings of the first ACM workshop on Secure Execution Untrusted Code (SecuCode’09)*, November 2009, pp 3-10.
- [28] N.Jovanovic, E.Kirda , C. Kruegel,”Preventing Cross-Site Request Forgery Attacks,”

- in *IEEE International Conference on Securecomm and Workshop* , August 2006, pp 1- 10.
- [29] www.webappsec.org/
- [30] S. Gupta,B. B. Gupta, ” Cross–site scripting(XSS) attacks and defense Mechanisms : classification and state-of-the-art, ” *International Journal of System Assurance Engineering and Management*, Springer , Vol 6, September 2015,pp1-19.
- [31] B.B. Gupta, S. Gupta, S.Gangwar, M.Kumar, P.K.Meena, ”Cross-site scripting (XSS) abuse and defense: exploitation on several testing bed environments and its defense,” *Journal of Information Privacy and Security*,Taylor & Francis , Vol. 11 , No.2,July 2015,pp 118–136.
- [32] A.Z.M Saleh,N.A.Rozali,A.G.Buja,K.A.Jalil,F.H.M. Ali,T.F.A. Rahman ,” A Method For Web Application Vulnerabilities Detection by Using Boyer - Moore String Matching Algorithm ,” *Procedia Computer Science* , Elsevier , ISSN: 1877- 0509, Vol.72,2015,pp112- 121.
- [33] B.B.Gupta, S .Gupta , ” XSS-SAFE: A Server-Side Approach to Detect and Mitigate Cross - Site Scripting (XSS) Attacks in JavaScript Code ,” *Arabian Journal for Science and Engineering*, Springer ,Vol. 41,No. 3, March 2016,pp 897-920.
- [34] M.I.P Salas, E. Martins ,” Security Testing Methodology for Vulnerabilities Detection Of XSS in Web Services and WS-Security ,” *Electronic Notes in Theoretical Computer Science*, Elsevier, ISSN 1571-0661, Vol. 302, February 2014,pp 133-154.
- [35] F. Duchene , R. Groz , S. Rawat , J.L. Richier , “ XSS Vulnerability Detection Using Model Inference Assisted Evolutionary Fuzzing*,” in *5th IEEE International Conference on Software Testing , Verification and Validation* , April 2012.
- [36] Y. Sun , D. He , “ Model Checking for the Defense against Cross-site Scripting Attacks ,” in *IEEE International Conference on Computer Science and Service*

System, August 2012.

- [37] L. K. Shar and H. B.K Tan , “ Mining Input Sanitization Patterns for Predicting SQL Injection and Cross Site Scripting Vulnerabilities”,in *34th IEEE International Conference of Software Engineering (ICSE)*, June 2012.
- [38] L.K. Shar and H.B.K. Tan, “Defending against Cross-Site Scripting Attacks,” *IEEE Computer Society* ,Vol. 45,No. 3, March 2012,pp 55-62.
- [39] A. Avancini ,M. Ceccato,“ Security Testing of Web Applications : a Search Based Approach for Cross - Site Scripting Vulnerabilities,” in *11th IEEE International Working Conference of Source Code Analysis and Manipulation (SCAM)* , September 2011.
- [40] R. Putthacharoen , P. Bunyatnoparat , “ Protecting Cookies from Cross Site Script Attacks Using Dynamic Cookies Rewriting Technique ,” in *13th IEEE International Conference of Advanced Communication Technology (ICACT)*, February 2011.
- [41] A. K. Zun , P. J. Guo , K. Jayaraman, M. D. Ernst, ”Automatic Creation of SQL Injection and Cross - Site Scripting Attacks , ” in *31st IEEE International Conference of Software Engineering (ICSE)*, May 2009, pp 199-209.
- [42] M.T. Louw, V.N. Venkatakrishnan , “ BLUEPRINT: Robust Prevention of Cross-Site Scripting Attacks for Existing Browsers ,” in *30th IEEE International Symposium on Security and Privacy*, May 2009,pp 331-346.
- [43] G.Wassermann , Z. Su , “ Static Detection of Cross -Site Scripting Vulnerabilities ,” in *30th ACM International Conference on Software engineering (ICSE'08)*, May 2008, pp 171-180.

- [44] J. Shanmugam , M. Ponnaivaikko , ” Behavior - based anomaly detection on the serverside to reduce the effectiveness of Cross Site Scripting vulnerabilities ,” in *3rd IEEE International Conference on Semantics , Knowledge and Grid*, October 2007.
- [45] Z. Qianjie,H Chen, J. San,”An Execution-flow based method for detecting cross-site scripting attacks , “ in *2nd IEEE Conference on Software Engineering and Data Mining(SEDM)*, June 2010, pp 160-165.
- [46] J. Shanmugam ,M. Ponnaivaikko , “Risk Mitigation for Cross Site Scripting Attacks Using Signature Based Model on the Server Side,” in *2nd IEEE International Multi - Symposiums on Computer and Computational Sciences (IMSCCS)*, August 2007.
- [47] J. Shanmugam, “ A solution to block Cross Site Scripting Vulnerabilities based on Service Oriented Architecture , ” in *6th IEEE International Conference on Computer and Information Science(ICIS)*, July 2007.
- [48] S. J. Wang ,Y. H.Chang , W. Y. Chiang and W.S. Juang, “Investigations in Cross Site Script on Web –systems Gathering Digital Evidence Against Cyber Intrusions,”*in Proceedings of Future Generation Communication and Networking (FGCN),IEEE Computer Society*,Vol.2, Dec 2007,pp 125-129.
- [49] P.Sharma , R. Johari ,S.S. Sarma ,”Integrated approach to prevent SQL injection attack and reflected cross site scripting attack,” in *International Journal of System Assurance Engineering and Management*, Springer ,Vol.3,No.4,pp 343- 351,2012.
- [50] G. Jiao ,C. M. XU,J.Msohua, “SQLIMW: a new mechanism against SQL-Injection ,“ in *IEEE International conference on Computer Science and Service System*, August 2012.

- [51] R. Dharam , S. G. Shiva , “ Runtime Monitors for Tautology based SQL Injection Attacks, ” in *IEEE International Conference on Cyber Security, Cyber Warfare and Digital Forensic (CyberSec)*, June 2012.
- [52] I. Balasundaram , E Ramaraj , “ An Authentication Scheme for Preventing SQL Injection Attack Using Hybrid Encryption ,” in *European Journal of Scientific Research*, Vol. 53, No. 3, 2011, pp 359-368.
- [53] S. Kunal, R.MohanDas, A.R.Pais, ” Model based hybrid approach to prevent SQL Injection attacks in PHP ,” in *1st International Springer conference on security aspects of information technology(InfoSecHiComNet'11)*, 2011, pp 3-15.
- [54] K.X. Zhang, C.J. Lin, S.J. Chen, Y. I.Hwang, H.L. Huang, F. H. Hsu, “ TransSQL: A Translation and Validation - based Solution for SQL - Injection Attacks, ” in *1st IEEE International Conference on Robot, Vision and Signal Processing (RVSP)* , November 2011.
- [55] A.Pomeroy, Q. Tan, "Effective SQL Injection Attack Reconstruction Using Network Recording, " in *11th IEEE International Conference in Computer and Information Technology (CIT)*, September 2011.
- [56] P. Bisht , P . Madhusudan , and V. N. Venkatakrishnan , " CANDID : Dynamic Candidate Evaluations for Automatic Prevention of SQL Injection Attacks," in *ACM Transactions on Information System Security*, Volume 13, No. 2, February 2010, pp 1–39.
- [57] M. Ruse , T. Sarkar and S . Basu , " Analysis & Detection of SQL Injection Vulnerabilities Via Automatic Test Case Generation of Programs," in *10th Annual International Symposium on Applications and the Internet*, July 2010, pp. 31 – 37.
- [58] X. Wang , L. Wang, G. Wei , D . Zhang , Y. Yang , ” Hidden Web Crawling

For SQL Injection Detection,”in *3rd IEEE International Conference on Broadband Network and Multimedia Technology (IC-BNMT)*,October 2010.

- [59] R. Ezumalai , G. Aghila , “ Combinatorial Approach for Preventing SQL Injection Attacks,” in *IEEE Conference on International Advance Computing Conference (IACC 2009)*, March 2009.
- [60] M. Junjin, “An Approach for SQL Injection Vulnerability Detection-AMNESIA,”in *6th International Conference on Information Technology:New Generations*, April 2009, pp 1411- 1414.
- [61] S.Thomas , L. Williams , T. Xie," On automated prepared statement Generation to remove SQL Injection vulnerabilities ," in *Information and Software Technology* ,ACM,Vol. 51, No. 3, March 2009.
- [62] M. Kiani , A. Clark and G. Mohay , ” Evaluation of Anomaly Based Character Distribution Models in the Detection of SQL Injection Attacks,” in *3rd IEEE International Conference on Availability, Reliability and Security (ARES)*, March 2008.
- [63] R. A. McClure and I. H. Krüger , “ SQL DOM : Compile Time Checking of Dynamic SQL Statements,” in *27th International ACM Conference on Software Engineering(ICSE)*,May 2005.
- [64] F. Valeur, D. Mutz, G. Vigna , " A Learning -Based Approach to the Detection of SQL Attacks,"in *2nd International ACM Conference on Detection of Intrusions and Malware and Vulnerability Assessment*, July 2005 ,pp 123-140.
- [65] C. Gould , Z. Su, P. Devanbu , “ JDBC Checker: A Static Analysis Tool for SQL/JDBC Applications,” in *26th International ACM Conference on Software Engineering (ICSE)*, May 2004, pp 697-698.

- [66] Y .Huang, S.Huang, T. Lin, C Tasi, " Web application security assessment by fault injection and behavior monitoring,"in12th *International ACM Conference on World Wide Web(WWW)*, May 2003,pp 148-159.
- [67] A. Barth, C.J.J.C. Mitchell , "Robust Defenses for Cross-Site Request Forgery," in *15th ACM Conference on Computer and Communications Security*, October 2008.
- [68] N. Jovanovic , E. Kirda, and C. Kruegel, "Preventing Cross Site Request Forgery Attacks , " in *IEEE International Conference on Securecomm and Workshops*, August 2006, pp 1-10.
- [69] M. S. Siddiqui , D. Verma , " Cross Site Request Forgery : A Web Common Application weakness, "in 3rd *IEEE International Conference on Communication Software and Networks (ICCSN)*, May 2011.
- [70] B.Chen , P. Zavorsky , R.Ruhl and D. Lindskog , " A Study of the Effectiveness of CSRF guard , " in 3rd *IEEE International Conference on Privacy, Security, Risk and Trust (PASSAT)*, October 2011.
- [71] W.Maes, T.Heyman, L. Desmet, W. Joosen , " Browser protection against cross site request forgery, " in 1st *ACM workshop on Secure Execution of Untrusted Code(SecuCode '09)*, November 2009.
- [72] T. Alexenko , M. Jenne, S.D. Roy ,W. Zeng , "Cross Site Request Forgery: Attack and Defense , " in 7th *IEEE International Conference on Consumer Communications and Networking Conference (CCNC)*, January 2010.
- [73] X.Lin, P. Zavorsky , R.Ruhl, D. Lindskog, "Threat modeling for CSRF Attacks," in *IEEE International Conference on Computational Science and Engineering*

(CSE),August 2009.

- [74] H. Shahriar and M. Zulkernine , “ Client side Detection of Cross Site request forgery attacks, ” in *21st IEEE International Symposium on Software Reliability Engineering (ISSRE)*, November 2010.
- [75] B. Nagpal, N.Chauhan , N.Singh , “ Cross – Site Request Forgery : Vulnerabilities and Defenses, ”in *I-Manager’s Journal on Information Technology(JIT)*,ISSN:2277-5110,Vol.3,No.2,2014,pp 13-21.
- [76] B. Nagpal, N.Chauhan, N.Singh,“SECSIX:Security Engine for CSRF,SQL Injection and XSS attacks, ” in *International Journal of System Assurance Engineering & Management (IJSA)* , Springer , ISSN - 0976-4348,2016.doi. 10.1007/s1 3198-016-0489-0.
- [77] B.Nagpal, N.Chauhan, N.Singh, ”A substitution based encoding scheme to mitigate Cross site script vulnerabilities , ” in *I - Manager’s Journal on Information Technology(JIT)*,ISSN: 2277-5110,Vol.5,No.1,2015,pp 7-12.
- [78] B.Nagpal ,N.Chauhan,N.Singh, ” Injection and Prevention of SQL Injection Attacks on Web Applications , ” in *International Journal of Software and Web Sciences (IJSWS)*, ISSN: 227 9-0071,Vol.2,No.9,2014,pp 125-128.
- [79] B. Nagpal, N.Chauhan, N. Singh,“A viable solution to prevent SQL injection attack using SQL injection, ”in *I - Manager’s Journal on Computer Science (JCOM)*, ISSN: 2347- 6141,Vol. 3,No.3,2015,pp 1-6.
- [80] B.Nagpal , N. Chauhan, N. Singh , “ Additional Authentication Technique : An Efficient approach to prevent cross site request forgery attack, ”in *I - Manager’s Journal on Information Technology (JIT)* , ISSN: 2277- 5110, Vol.5,

No.2,2016,pp.14-18.

- [81] S.Som, S.Sinha, R. Kataria , “ Study on SQL injection attacks:Mode,Detection and Prevention” ,in *International Journal of Engineering Applied Sciences and Technology(IJEAST)*,ISSN:2455-2143,Vol. 1,No.8,2016,pp 23-29.
- [82] M. S. S Sayyad, T.P. Bahare , “Study of SQL Injection Attacks ,” in *International Journal of Computer and Communication Engineering* , Vol. 2 ,No.5,Sept. 2013.
- [83] R. P. Mahapatra , S. Khan , “ A Survey of SQL Injection Countermeasures, ” in *International Journal of Computer Science & Engineering Survey (IJCSSES)* , Vol. 3, No. 3,June2012.
- [84] K. Elshazly, Y.Fouad , M .Saleh , A.Sewisy, “ A Survey of SQL Injection Attack Detection and Prevention, ” in *Journal of Computer & Communications* , Vol. 204,No.2,2014,pp1-9.
- [85] M .Rohilla , R. Kumar, G. Gopal , “ XSS Attacks : Analysis , Prevention and Detection , ” in *International Journal of Advanced Research in Computer Science & Software Engineering (IJARCSSE)*, ISSN: 2277-128X,Vol.6, No.6, June 2016.
- [86] A.Shrivatava,S.Chaudhary,A.Kumar,“XSS vulnerability assessment and prevention in web application, ” in *2nd IEEE International Conference on Next Generation Computing Technologies(NGCT)*,Oct.2016.
- [87] M.V.Panah, N.K.Bayat,A.Asami,M.A. Shahmirzadi,“ SQL Injection Attacks : A Systematic Review, ”in *International Journal of Computer Science & Information Security(IJCSIS)*,ISSN:1947-5500, Vol. 14,No.12,Dec.2016,pp 678-696.
- [88] R.D Giri , P.S. Kumar , L. Prasannakumar , V.R.N.V Murthy , “ Object Oriented Approach to SQL Injection Preventer, ” in *3rd International Conference on*

Computing Communication & Networking Technologies(ICCNT),2012.

- [89] M. Sonoda, T. Matsuda, D. Koizumi , S. Hirasawa ,“On automatic detection of SQL Injection Attacks by the feature extraction of the single character ,” in *4th International Conference on Security of Information & Networks(SIN'11)*,2011.
- [90] N. Manaswini ,P.K. Sahoo , “ CSRF attacks on web applications, ” in *International Journal of Advanced Computing Technique and Applications (IJACTA)*, ISSN: 2321-4546,Vol.4,No.1,June2016.

BRIEF PROFILE OF RESEARCH SCHOLAR

Ms. Bharti Nagpal did her M.Tech(Information Systems) from N.S.I.T Dwarka, Delhi in 2009 and B.Tech(Computer Engineering) from N.I.T. Kurukshetra in 1999. She has over 16 years of experience in teaching in B.Tech and MCA courses. Her areas of interest includes Information Security, Operating Systems, Programming Languages , Web Technologies. She has published 30 research papers in various journals and conferences of international fame. Currently, she is working as Assistant Professor in the Department of Computer Engineering at Ambedkar Institute of Advanced Communication Technology & Research(AIACT&R), Delhi.

LIST OF PUBLICATIONS OUT OF THESIS

(i)List of Published Papers (International Journal)

S.No	Title of Paper	Name of Journal where published	No.	Volume & Issue	Year	Pages
1.	Approaches to Detect and Prevent Cross-Site Scripting Attacks on websites : A Survey	I-manager Journal on Information Technology (JIT)	2277-5110	Vol. 2,4	2013	36-43
2.	Cross-Site Request Forgery: Vulnerabilities and Defenses	I-manager Journal on Information Technology (JIT)	2277-5110	Vol.3,2	2014	13-21
3.	A Survey on the Detection of SQL Injection Attacks and Their Countermeasures	Journal of Information Processing Systems (JIPS),Korea (ESCI)	1976-913X	Vol.13,4	2017	689-702
4.	Injection and Prevention of SQL Injection Attacks on Web Applications	International Journal of Software and Web Sciences (IJSWS)	2279-0071	Vol. 2,9	2014	125-128
5.	Defending against Remote File Inclusion attacks on web applications	I- manager Journal on Information Technology (JIT)	2277-5110	Vol.4,3	2015	25-33
6.	A viable solution to prevent SQL Injection attack using SQL Injection	I-manager Journal on Computer Science (JCOM)	2347-6141	Vol.3,3	2015	1-6
7.	A substitution based encoding scheme to mitigate Cross Site Script vulnerabilities.	I-manager Journal on Information Technology (JIT)	2277-5110	Vol.5,1	2015	7-12
8.	Additional Authentication	I-manager Journal on	2277-5110	Vol.5,2	2016	14-18

	Technique: An efficient approach to prevent Cross Site Request Forgery attack.	Information Technology (JIT)				
9.	SECSIX: Security Engine for CSRF,SQL Injection and XSS attacks.	International Journal of System Assurance Engineering & Management (IJSA), Springer (<i>SCOPUS</i>)	0976-4348	doi 10.1007/s13198-016-0489-0	2016	

(ii)List of Communicated Papers

S. No	Title of the Paper	Name of Journal	Present Status	Year
1.	Syntax analysis based hybrid approach to mitigate web attacks	International Arab Journal of Information Technology(IAJIT) (<i>SCIE</i>)	Under review	2017

LIST OF RESEARCH PAPERS

International Journal

- [1] Nagpal Bharti , Chauhan Naresh , Singh Nanhay , "Approaches to Detect and Prevent Cross – Site Scripting Attacks on Websites : A Survey, ” *I-Manager Journal on Information Technology(JIT)*,ISSN :2277-5110,Vol.2,No.4, 2013, pp 36-43.
- [2] Nagpal Bharti, Chauhan Naresh, Singh Nanhay , " A Survey on the Detection of SQL Injection Attacks and Their Countermeasures ,” *Journal of Information Processing Systems(JIPS)*,Korea,ISSN: 1976-913X, Vol.13, No.4, 2017, pp 689-702.
- [3] Nagpal Bharti , Chauhan Naresh , Singh Nanhay , “ Cross – Site Request Forgery : Vulnerabilities and Defenses, ” *I-Manager Journal on Information Technology (JIT)*, ISSN: 2277- 5110,Vol.3, No.2, 2014, pp 13-21.
- [4] Nagpal Bharti , Chauhan Naresh , Singh Nanhay , “ SECSIX : Security Engine for CSRF, SQL Injection And XSS attacks, ” *International Journal of System Assurance Engineering & Management (IJSA)* , Springer , ISSN - 0976-4348 ,doi. 10.1007/s13198-016- 0489-0, 2016.
- [5] Nagpal Bharti , Chauhan Naresh , Singh Nanhay , ” A substitution based encoding scheme to mitigate Cross site script vulnerabilities , ” *I - Manager Journal on Information Technology(JIT)*,ISSN: 2277-5110,Vol.5,No.1, 2015, pp 7-12.
- [6] Nagpal Bharti , Chauhan Naresh , Singh Nanhay , ” Injection and Prevention of SQL Injection Attacks on Web Applications, ” *International Journal of Software and Web Sciences (IJSWS)*, ISSN: 2279-0071, Vol. 2, No.9, 2014, pp 125-128.

- [7] Nagpal Bharti , Chauhan Naresh , Singh Nanhay , “ A viable solution to prevent SQL injection attack using SQL injection, ” *I-Manager Journal on Computer Science (JCOM)*,ISSN: 2347- 6141,Vol. 3,No.3, 2015, pp 1-6.
- [8] Nagpal Bharti , Chauhan Naresh , Singh Nanhay , “ Defending against Remote File Inclusion attacks on web applications, ” *I-Manager Journal on Information Technology (JIT)* ,ISSN- 2277-5110,Vol.4,No.3,2015, pp.25-33.
- [9] Nagpal Bharti , Chauhan Naresh , Singh Nanhay , “ Additional Authentication Technique : An Efficient approach to prevent cross site request forgery Attack, ” *I-Manager Journal on Information Technology (JIT)* , ISSN: 2277-5110, Vol. 5, No. 2, 2016, pp.14-18.

International Conference

- [1] Nagpal Bharti , Singh Nanhay , Chauhan Naresh , Panesar Angel , “ Tool based Implementation of SQL Injection for Penetration Testing, ” in *IEEE International Conference on Computing, Communication and Automation (ICCCA-2015)*,2015.
- [2] Nagpal Bharti , Chauhan Naresh , Singh Nanhay , Sharma Pratima , “ CATCH : Comparison and Analysis of Tools covering Honeypot , ” in *IEEE International Conference on Advances in Computer Engineering and Applications (ICACEA-2015)* ,2015.
- [3] Nagpal Bharti , Chauhan Naresh , Singh Nanhay , Kamal Pratibha , “ Analysis and Comparison of Web Application Firewall Tools , ” in *IEEE International Conference on Advances in Computer Engineering and Applications (ICACEA-2015)*,2015.

- [4] Nagpal Bharti ,Chauhan Naresh , Singh Nanhay , Sharma Pratima , “ Preventive Measures For Securing Web Applications Using Broken Authentication And Session Management Attacks: A Study, ”in *International Conference on Advances in Computer Engineering and Applications (ICACEA-2014)*,2014.
- [5] Nagpal Bharti ,Chauhan Naresh , Singh Nanhay , Murari Radhika , “A Survey and Taxonomy of Packet Classification algorithms, ” in *IEEE International Conference on Advances in Computer Engineering and Applications (ICACEA-2015)*,2015.