Approaches to detect SQL injection and XSS in web applications

Abhishek Kumar Baranwal
Masters of Software Systems
University of British Columbia
657, 57th Avenue East
Vancouver, Canada
burnee@mss.icics.ubc.ca

ABSTRACT
We are increasingly relying on web, and accessing important information as well as transmitting data through it. At the same time, quantity and impact of security vulnerabilities in such applications has grown as well. Billions of transactions are performed online with the help of various kinds of web applications. Almost in all of them user is authenticated before providing access to backend database for storing all the information. In this whole scenario a well-designed injection can provide access to malicious or unauthorized users and mostly achieved through SQL injection and Cross-site scripting (XSS). In this paper we are going to provide a detailed survey of various kinds of SQL injection, XSS attacks and approaches to detect and prevent them. Furthermore we are also going to provide a comparative analysis of different approaches against these attacks. And then we are also going to present our findings and note down future expectations and expected development of counter measures against these attacks.

Keywords
SQL injection attacks; Prevention; Detection; Cross-site scripting (XSS); XSS attacks; Techniques; Evaluation; Web applications; Fault injection;

1. INTRODUCTION
In recent years the development of internet had huge impact on commerce and culture. As each coin has two sides; The World Wide Web has both pros and cons. On one hand where the Web can dramatically lower costs to organizations for distributing information, products, and services on the other computers that make up the Web are vulnerable. Security was, still is, and always will be one of the major concerns that critical systems have, especially when deployed in the World Wide Web, accessible through a web browser. Most security concerns are now related with application level. Today almost everyone in touch of computer is connected online and to serve millions of users huge amount of data is stored in databases connected to some web application all across the globe. These users keep on inserting, querying and updating data in these databases. For all these operations to occur successfully a well-designed user interface is very important. Furthermore, these applications play a vital role in maintaining security of data stored in these databases. Web applications which are not very secured allow well-designed injection to perform unwanted operations on backend database. An unauthorized user may be able to exploit this situation by damaging or with theft of trusted users sensitive data stored here. The maximum damage is caused when an attacker gains full control over a database or web application as there is a chance of system being fully destroyed. The sensitive data on one's system could be transferred to any third party by injecting a class of vulnerability in trusted sites' dynamically generated pages. This third party could be an attacker's server. This mechanism could also be used in avoiding cookie protection mechanism or same-origin-policy. SQL injection and XSS are two major kinds of attack which has been used frequently to serve this purpose. SQL Injection is a method of injecting SQL meta-characters or commands inside Web-based input fields so that execution of the back-end SQL queries can be manipulated. Web servers belonging to an organization is primary target of these kinds of attack. XSS attack comes into play by embedding script tags in URLs and attracting unsuspecting or trusted users to click on them. It is being ensured that the malicious Script is executed on the victim's local machine. These attacks take the benefit of trust existing between the user and the server. They also take advantage of absence of input/output validation on the server to reject malicious JavaScript characters [59]. XSS usually affects victim's web browser on the client-side where as SQL injection related web vulnerability is involved with server side. These vulnerabilities could be exploited by SQL injection or XSS to gain control over the online web application database or stealing users information. In this paper we have put focus on all kinds of SQL injection and XSS attacks and approaches for their detection and prevention.

In fact, SQL Injection and XSS are categorized as the top-10 2010 Web application vulnerabilities experienced by Web applications according to OWASP (Open Web Application Security Project) project [4].

2. DISCUSSION
The organization of the paper will start by briefly discussing SQL Injection in section 3 and its different types in section 3.1. In section 3.2 we discuss various approaches suggested by different authors to detect SQL injection, their workings and limitations. Then in section 3.3 an analytical evaluation of different approaches to detect SQL injection has been done against different kinds of injection. In section 4 we will put our focus on briefly discussing Cross -Site Scripting attack, its different kinds till known so far will be discussed in section 4.1. Then in section 4.2 we will see working of approaches developed to detect XSS in web applications.

Section 4.3 contains evaluation of approaches suggested by different authors against existing XSS attacks. Finally in section 5 this paper is concluded and there will be a brief discussion on new trends and direction.

3. SQL INJECTION

SQL injection attack is a kind of attack where an attacker sends SQL (Structure Query Language) code to a user input box in a web form of a web application to gain unlimited and unauthorized access. The attacker's input is transmitted into an SQL query in such a way that it forms an SQL code [2], [3].

3.1 SQL Injection Types

These are the classification of SQL injection types according to Halfond, Viegas and Orso researches [1], [5]:-

3.1.1 Tautology:

This attack bypasses the authentication and access data through vulnerable input field using "**where"** clause by injecting SQL tokens into conditional query statements which always evaluates to true.

Example – *Select * from <tablename> where userId = <id> and password = <wrongPassword> or 1=1;*

3.1.2 Logically incorrect queries:

The error message sent from database on being sending wrong SQL query may contain some useful debugging information. This could help in finding parameters which are vulnerable in the web application and hence in the database of the application

Example.- The error message for sending a wrong password may be like:- *Select * from <tablename> where userId = <id> and password = <wrongPassword> or 1=1;*

From this information the attacker is likely to know the table name and name of the fields in the database which could be used further to prepare a more organized attack.

3.1.3 Union queries:

The "Union" keyword in SQL can be used to get information about other tables in the database. And if used properly this can be exploited by attacker to get valuable data about a user from the database.

Example- Select * from <tablename> where userId = <id> and password = *<rightPassword> Union select creditCardNumber from CreditCardTable;*

The part in italics in the query is sent by attacker as input to password textbox.

3.1.4 Piggy-backed Queries:

This is the kind of attack where an attacker appends ";" and a query which can be executed on the database. It could be one of the very dangerous attacks on database which could damage or may completely destroy a table. If this attack is successful then there could be huge loss of data.

Example- Select * from <tablename> where userId = <id> and password = *<rightPassword>; Drop table <tablename>;*

The part in italics in the query is sent by attacker as input to password textbox.

3.1.5 Stored Procedure:

It is an abstraction layer on top of database and depending on the kind of stored procedure there are different ways to attack. The vulnerability here is same as in web applications. Moreover all the types of SQL injection applicable for a web application are also going to work here.

3.1.6 Blind Injection:

It's difficult for an attacker to get information about a database when developers hide the error message coming from the database and send a user to a generic error displaying page. It's at this point when an attacker can send a set of true/false questions to steal data.

Example- *SELECT name FROM <tablename> WHERE id=<username> and 1 =0 -- AND pass =*
SELECT name FROM <tablename> WHERE id=<username> and 1 = 1 -- AND pass =

Both the queries will return an error message in case the web application is secure, however if there is no validation for input then the chances of injection exist. If attacker receives an error after submitting the first query, he might not know that, was it because of input validation or error in query formation. After that on submission of the second query which is always true if there is no error message then it clearly states that id field is vulnerable.

3.1.7 Timing Attacks:

In this kind of attack timing delays are observed in response from a database which helps to gather information from a database. SQL engine is caused to execute a long running query or a time delay statement with the help of if-then statement which depends on the logic that has been injected. It is possible to determine whether injected statement was true or false depending on how much time page takes to load. The keyword WAITFOR along the branches can cause response delay for a given time in a database.

Example- Declare @s varchar(500) select @s = db_nameO if (ascii(substring(@s, I, I)) & (power(3, 0))) > O waitfor delay '0:0:10'

In this example database gets paused for ten seconds if in the database used, the first bit of the first byte of the name is 1.So, when condition is true this code is injected to produce response delay in time.

3.1.8 Alternate Encodings:

This technique is used to modify injection query by using alternate encodings, like – Unicode, ASCII, hexadecimal. In this way attacker can escape the filter for "wrong characters". It could be dangerous if used in combination with other techniques as it can target different layers of a web application. All different kinds of SQL injection attack can be hidden using this method through alternate encodings.

Example- *SELECT name FROM <tablename> WHERE id='' and password=O;exec(char(O x73687574646j776e))*

The actual character is returned by the char function used here which takes hexadecimal encoded characters as input. During execution this encoded string gets converted into shutdown command for database.

3.2 Approaches to Detect SQL Injection

A web application can be protected from SQL injection attack by taking into account two major things:-

I. Mechanism to detect SQL injection attacks.
II. Knowledge of SQL injection vulnerabilities in web applications.

Here we are going to look at most prominent solutions and their workings in brief and know about core ideas behind each of them.

3.2.1 Shin et al.'s Approach

SQLUnitGen a tool based on static analysis has been suggested to identify input vulnerabilities manipulation. In Table 1, a comparison has been done between this tool and FindBugs which is again a tool based on static analysis [6]. The mechanism proposed here has been effective on the fact that no false positive were found even after conducting an experiment with 483 attack test cases. Furthermore in different scenarios only a small number of false negatives were noticed. In addition for some of the applications a more significant number of false negatives may occur due to some shortcomings. So, the author has talked about bringing an improvement in the tool in future and getting rid of these shortcomings leading to significant false negatives [6].

3.2.2 Database Security Testing Approach by Haixia and Zhihong

A secured database testing approach has been suggested for web applications by Haixia and Zhihong.

This approach suggested following:-

I. Detection of potential input points of SQL injection.
II. Automatic generation of test cases.
III. Running the test cases to make an attack on the application to find the database vulnerability.

The mechanism suggested here is shown to be efficient as it was able to detect the input points of SQL Injection exactly and on time as per expectation. An analysis on this technique makes it clear that the approach needs improvement in the development of attack rule library and detection capability [7].

3.2.3 SAFELI

Fu et al. suggested a Static Analysis approach to detect SQL Injection Vulnerabilities. The main aim of SAFELI approach is to identify the SQL Injection attacks during compile-time. It has a couple of advantages. First, it performs a White-box Static Analysis and second, it uses a Hybrid-Constraint Solver. On one hand where the given approach considers the byte-code and deals mainly with strings in case of White-box Static Analysis, on the other through Hybrid-Constraint Solver, the method implements an efficient string analysis tool which is able to deal with Boolean, integer and string variables. Its implementation was done on ASP.NET Web applications and it was able to detect vulnerabilities that were ignored by the black-box vulnerability scanners. This approach is an efficient approximation mechanism to deal with string constraints. However, the approach is only dedicated to ASP.NET vulnerabilities [8].

3.2.4 Ruse et al.'s Approach

The approach suggested here uses automatic test case generation for detection of SQL injection vulnerabilities. The idea is to create a specific model that deals with SQL queries automatically. Furthermore this technique also identifies the relation between sub-queries. This technique is shown to identify the casual set and obtain 85% and 69% reduction respectively while experimenting on few sample examples. Moreover, no false positive or false negative were produced and it has been able to detect the root cause of the injection.

Table 1: Comparison with static analysis tool

Applications	Tools	Vulnerable hotspots	Vulner-abilities found	False positives	False negatives
Bookstore 1	SQLUnitGen	1	1	0 (0%)	0
	FindBugs	1	1	0 (0%)	0
Bookstore 2	SQLUnitGen	1	1	0 (0%)	0
	FindBugs	1	1	0 (0%)	0
Bookstore 3	SQLUnitGen	0	0	0 (0%)	0
	FindBugs	0	1	1 (100%)	0
Cabinet 1	SQLUnitGen	5	3	0 (0%)	2
	FindBugs	5	5	0 (0%)	0
Cabinet 2	SQLUnitGen	1	1	0 (0%)	0
	FindBugs	1	5	4 (80%)	0
Cabinet 3	SQLUnitGen	0	0	0 (0%)	0
	FindBugs	0	5	5 (100%)	0

Although this approach claimed an apparent efficiency, it has a huge drawback that this approach has not been tested on real life existing database with real queries [9].

3.2.5 Thomas et al.'s Approach

This approach suggested an automated prepared statement generation algorithm to eliminate vulnerabilities related to SQL Injection. Their research work used four open source projects namely: (i) Net-trust, (ii) ITrust, (iii) WebGoat, and (iv) Roller. The experimental results show that, their prepared statement code was able to successfully replace 94% of the SQL injection vulnerabilities in four open source projects. The only limitation observed was that the experiment was conducted using only Java with a limited number of projects. Hence, the wide application of the same approach and tool for different settings still remains an open research issue to investigate [10].

3.2.6 Roichman and Gudes's Fine-grained Access Control Approach

The approach suggested here for securing web applications uses fine-grained access control to web databases. The built in database control is used to supervise and monitor access to corresponding database. This mitigates the risk of attack at database application backend because of the fact that security and access control of a database is transferred from the application layer to the database layer. SQL session traceability vulnerability has this solution. Furthermore it is a framework which applies to all kinds of database application [11]. Table 2 shows the performance evaluation chart for it [11].

3.2.7 SQL-IDS Approach

This approach has been suggested by Kemalis and Tzouramanis in [12] and it uses a novel specification-based methodology for detecting exploitations of SQL injection vulnerabilities. The method proposed here does query-specific detection which allows the system to perform concentrated analysis at almost no computational overhead. It also does not produce any false positives or false negatives. This is a very new approach and in practice it's very efficient; however, it is required to conduct more experiments and do comparison with other detection methods under a flexible and shared environment.

3.2.8 AMNESIA

Junjin proposed AMNESIA which is an approach for tracing SQL input flow and generating attack input. JCrasher has been used for generating test cases, and SQLInjectionGen to identify the hotspots. Two web applications were used for conducting this experiment on MySQL1 1 v5.0.21. SQLInjectionGen produced only two false negatives in three attempts on two different databases. This framework has been effective on the fact that it emphasized on attack input precision. Furthermore attack input is matched properly with arguments of the method. The most important advantage of this approach is that there are no false positives and a very small number of false negatives.

The only disadvantage of this technique is that it has a number of steps involved using different tools [13]

3.2.9 SQLrand Approach

SQLrand is an approach proposed by Boyd and Keromytis in which randomized SQL query language is used, pointing a particular CGI (Common Gateway Interface) in an application. A proxy server was used in between the SQL server and Web server for implementation; the queries received from the client were de-randomized and request was sent to the server. This technique has two main advantages: security and portability. The proposed approach has a very good performance: every query has maximum latency overhead of 6.5 milliseconds. So, considering the performance obtained and strong defense against injected queries it is a very efficient approach. However, as this is a proof of concept only; A lot of support is required from programmers in building tools using SQLrand to target more DBMS back-ends for further testing [14].

3.2.10 SQL DOM Approach

McClure and Krüger suggested a framework SQL DOM (strongly-typed set of classes with database schema). The existing flaws have been considered closely during access of relational databases from Object-Oriented Programming Language's point of view. The focus lies mainly in identifying hurdles in interacting with databases through Call Level Interfaces. The solution proposed here is based on SQL DOM object model to handle this kind of issues by creating a secure surrounding i.e., creation of SQL statement through object manipulation for communication. When this technique was evaluated qualitatively it showed many

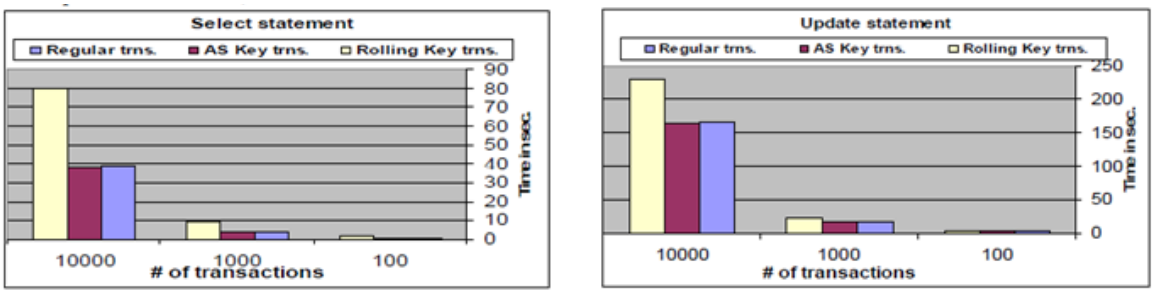

Table 2. Performance Evaluation Chart of Fine Grained Access Control Approach

advantages for: testability, readability, maintainability and error detection at compile. Although this proposal is efficient, there still exists scope of further improvements with latest and more advanced tools such as CodeSmith [15, 16].

3.2.11 Using Stored Procedures

Stored procedures are functions in the database which can be called by an application to do various operations on a database tables. A combination of static and runtime analysis is used in the detection and prevention in these stored procedures. Stored procedure parser is used for identifying the commands and SQLChecker is used at runtime for identifying the input. Huang et al. proposed a combination of static and runtime analysis and monitoring to bolster the security of potential vulnerabilities. Web application Security by Static Analysis and Runtime Inspection (WebSSARI) was used and implemented on 230 open source applications on SourceForge.net. The approach was found to be effective although it failed to remove the SQL Injection Vulnerabilities. It was only able to list the input either white or black [31, 26].

3.2.12 Parse Tree Validation Approach

A parser tree framework was adopted by Buehrer et al. . The original statement was compared with parsed tree of a particular statement dynamically at runtime. The execution of the statement was stopped unless there was a match found. A student's web application was used for this method using SQLGuard. Although the technique was found out to be efficient, it contained two major deficiencies: listing of input only and additional overheard computation [33].

3.2.13 Dynamic Candidate Evaluations Approach

An approach has been proposed by Bisht et al. called CANDID (Candidate evaluation for Discovering Intent Dynamically). It is a Dynamic Candidate Evaluations technique in which SQL injection is not only detected but also prevented automatically. Mechanism behind this method is that it dynamically extracts the query structures from every SQL query location which is intended by the developer. So, basically it resolves the problem of manually changing the application to produce the prepared statements. Although tool using this mechanism has been shown to be efficient in some cases, it failed for many other cases. An example for its failure is when applied at a wrong level or when an external function is dealt with. Furthermore it also fails in many cases due to limited capability of this technique [34].

3.2.14 Ali et al.'s Approach

This approach has been adopted by Ali et al. in which a hash value technique has been followed to improve user authentication mechanism. Hash values for user name and password has been used. For testing this kind of framework SQLIPA (SQL Injection Protector for Authentication) was developed. Hash values for user name and password is created for the first time user account is created and they stored in the user account table in a database.

The framework requires further improvement in order to minimize the overhead time which was 1.3 ms even though tested on few sample data. Hence simply minimizing the overhead time is not sufficient but also to test this framework with large amount of data is required [35].

3.2.15 SQLCHECKER Approach

Su and Wassermann implemented their algorithm with SQLCHECK on a real time environment. It checks whether the input queries conform to the expected ones defined by the programmer. A secret key is applied for the user input delimitation. The analysis of SQLCHECK shows no false positives or false negatives. Also, the overhead runtime rate is very low and can be implemented directly in many other Web applications using different languages. Table 3 shows the number of attacks attempted as well as prevented [36]. It also shows the number of valid uses attempted and allowed, and the mean and standard deviation of times across all runs of SQLCHECK for the application under check. It is a very efficient approach; however, once an attacker discovers the key, it becomes vulnerable. Furthermore, it also needs to be tested with online Web applications [36, 3].

Table 3. Precision and timing results for SQLCHECK

Language	Subject	Queries		Timing (ms)	
		Legitimate (Attempted/allowed)	Attacks (Attempted/prevented)	Mean	Std Dev
PHP	Employee Directory	660 / 660	3937 / 3937	3.230	2.080
	Events	900 / 900	3605 / 3605	2.613	0.961
	Classifieds	576 / 576	3724 / 3724	2.478	1.049
	Portal	1080 / 1080	3685 / 3685	3.788	3.233
	Bookstore	608 / 608	3473 / 3473	2.806	1.625
JSP	Employee Directory	660 / 660	3937 / 3937	3.186	0.652
	Events	900 / 900	3605 / 3605	3.368	0.710
	Classifieds	576 / 576	3724 / 3724	3.134	0.548
	Portal	1080 / 1080	3685 / 3685	3.063	0.441
	Bookstore	608 / 608	3473 / 3473	2.897	0.257

3.2.16 Detecting Intrusions in Web Databases (DIWeDa) Approach

Intrusion Detection Systems has been proposed by Roichman and Gudes for the backend databases. DIWeDa which is a prototype and acts at the session level rather than the SQL statement or transaction stage has been used by them for detection of intrusions in Web applications. In a particular session, DIWeDa determines the normal behavior of different roles in terms of the set of SQL queries issued, and then does a comparison with one having the profile to identify intrusions. The proposed framework is efficient and could easily identify SQL injections as well as business logic violations. However, with a threshold of 0.07, the True Positive Rate (TPR) was found to be 92.5% and the False Positive Rate (FPR) was 5%. Hence, there is a great need of accuracy improvement (Increase of TPR and decrease of FPR). It also needs to be tested against new types of Web attacks [37].

3.2.17 Manual Approach

The use of manual approach [13] has been highlighted by MeiJunjin in order to prevent SQL Injection input manipulation flaws. Defensive programming and code reviews are two main methods followed for application of manual approach. An input filter implemented in defensive programming restricts user to input malicious characters and keywords. White lists and Black lists are mainly used to achieve this objective. If we compare both the mechanism then in comparison to the code review [38], it is a low cost mechanism in detecting bugs; however, it requires deep knowledge on SQL Injection Attacks.

3.3 Comparative Analysis for SQL Injection

Every approach has some benefits depending on the settings of the system configured, so it would not be easy to get an idea about which one is the best. In Table 4 we show a chart of different approaches against different kinds of SQL injection attacks. Table 4 also shows a comparative analysis of SQL injection detection and prevention techniques with attack types.

Table:4 Various Approaches of SQL Injection and their Attacks

Attacks/ Approaches	Tautology	Logically Incorrect Queries	Union Query	Piggy-Backed Queries	Stored Procedure	Blind Injection	Timing Attacks	Alternate Encoding
AMNESIA	**Yes**	**Yes**	**Yes**	**Yes**	**No**	**Yes**	**Yes**	**Yes**
CANDID	**Yes**	**No**	**No**	**No**	**No**	**No**	**No**	**No**
DIWed	**No**	**No**	**No**	**No**	**No**	**Yes**	**Yes**	**No**
SQLrand	**Yes**	**No**	**Yes**	**Yes**	**No**	**Yes**	**Yes**	**No**
SQLCHECK	**Yes**	**Yes**	**Yes**	**Yes**	**No**	**Yes**	**Yes**	**Yes**
SQLDOM	**Yes**	**Yes**	**Yes**	**Yes**	**No**	**Yes**	**Yes**	**Yes**
SQLGuard	**Yes**	**Yes**	**Yes**	**Yes**	**No**	**Yes**	**Yes**	**Yes**
SQLIPA	**Yes**	**No**	**No**	**No**	**No**	**No**	**No**	**No**
WebSSARI	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**

Table 5: Objective of Various Approaches

Approaches	Goals	
	Detection	**Prevention**
AMNESIA	**Yes**	**Yes**
CANDID	**Yes**	**No**
DIWeDa	**Yes**	**No**
SQLrand	**Yes**	**Yes**
SQLCHECK	**Yes**	**No**
SQL DOM	**Yes**	**Yes**
SQLGuard	**Yes**	**No**
SQL-IDS	**Yes**	**Yes**
SQLIPA	**Yes**	**No**
WebSSARI	**Yes**	**Yes**

Although there exist many approaches to identify and prevent these attacks, only a few of them have been implemented practically. Hence, this comparison is based on analytical evaluation rather than empirical experience.

In Table 5, we have shown the major approaches to combat with SQL Injection and classified them on the basis of their features.

4. CROSS-SITE SCRIPTING (XSS)

Cross-Site Scripting (XSS) are the kinds of attacks on the web applications where an attacker is able to get control of user's browser for executing a malicious script which is mainly in the form of HTML/JavaScript code. It mainly lies in the context of trust of the web application's site. In result to it an attacker may be able to reach any sensitive browser resource related to the web application (passively or actively) if in case the embedded code gets executed successfully. Examples - cookies, session IDs, etc. [17].This problem has been figured out in two fold in [24]. Firstly, by design the browsers are not secure. They have been created with the motive of producing outputs with respect to a request and it is not considered the main duty of a browser to figure out whether a piece of code is doing something malicious. Secondly, because of lack in programming knacks and timeline constraint web application developers are unable to create secure sites.

4.1 Types of XSS attacks

XSS attacks are mainly of three types: persistent attacks, non-persistent attacks and DOM based [23].

4.1.1 Non-Persistent Attack

This is most common type of XSS vulnerability. The malicious code is not stored persistently; however it is reflected to the user immediately. As for example, let us consider a form for search query which displays results on to the page and the query is not being filtered for any scripting code. This can be easily exploited by an attacker through sending a victim user an email containing special designed link which is pointing to this search form and contains a malicious code in JavaScript. If the victim gets tricked and clicks this link then search form gets submitted with JavaScript code as query string, as a result the Script is sent back to the user victimized, as a part of the web page with result. As soon as this script gets executed, the cookie set by valid site will be forwarded to the malicious web site in the form of parameter to the invocation of the steal-<page> server-side script. This cookie will get saved on the malicious website. This can be further manipulated by attacker through impersonating the unsuspecting user in respect to trusted site [21, 22]. A sample scenario of non-persistent attack is explained in the Figure 1 [17].

4.1.2 Persistent Attack

This is a type of XSS attack where malicious code is stored in the persistently in a resource like file system, database or some other location which is managed by server and displayed to the users later. There is even no requirement of encoding as it is sent using html entities. As for example in case of an online message board users post messages which can be accessed by others later on some time [22].

Similarly as Figure1, a sample scenario of persistent attack is explained in the Figure2 [17].

4.1.3 DOM based XSS Attack

This kind of XSS attack is performed by changing the DOM environment on client side instead of sending the malicious code to server. So there is no chance of verification of payload by the server. In this kind of invasion major browsers are compelled to not to send the malicious payload to server. As a result even well-setup XSS filters become null against such attacks.

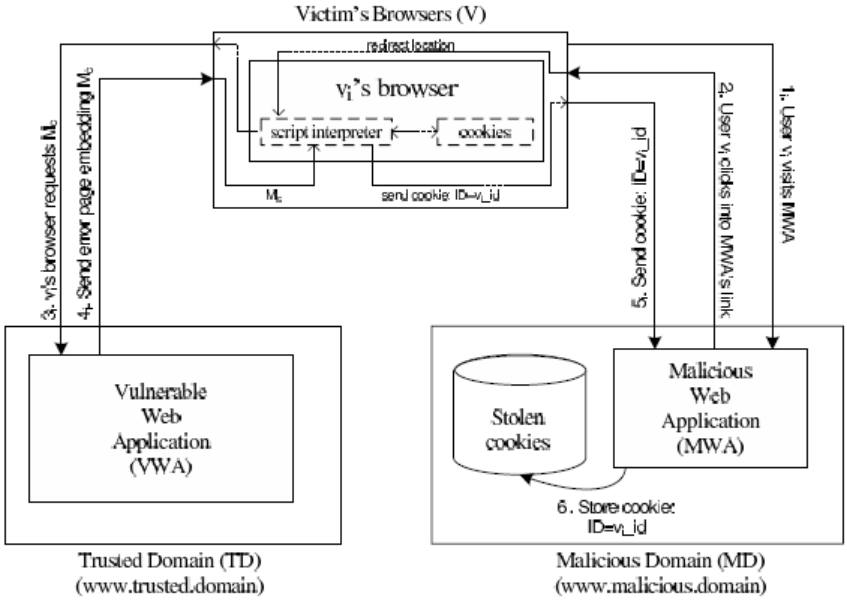

Figure1: Non-Persistent XSS attack sample scenario

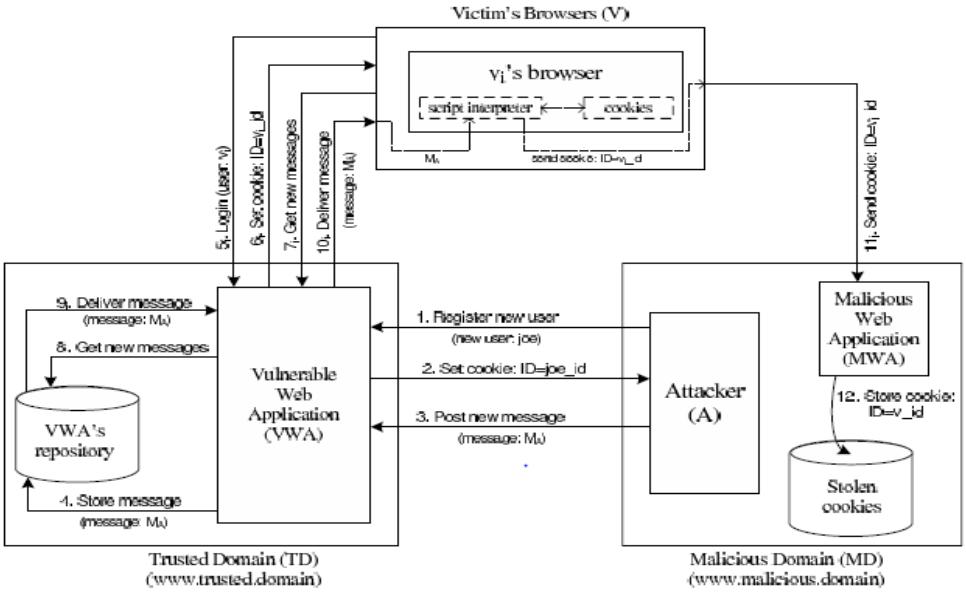

Figure2: Persistent XSS attack sample scenario

4.2 Approaches to Detect XSS

There are mainly three kinds of approaches followed since so far to detect XSS in web application. They are: Static Analysis, Dynamic Analysis and Combination of Static and Dynamic Analysis.

4.2.1 Static Analysis approach

Bounded Model Checking:

Counterexample traces has been used by Huang et al. to minimize the number of sanitization routines inserted and to identify the reason of errors that enhance the precision of both code instrumentation and error reports [40]. Variables representing current trust level were assigned states which further were used in verifying the legal information flow in a web application. Now in order to verify the correctness of all safety states of program Abstract Interpretation, Bounded Model Checking technique was used. Some of the major problems in their approach was to leave out alias analysis or include file resolution issues [29].

Analysis of String:

This approach has been suggested by Christensen, Mǿller, and Schwartzbach in the form of study of static string analysis for imperative languages. They have shown usefulness of string in checking for errors in dynamically generated SQL queries and analysis for analyzing reflective code in Java programs [42]. They used finite state automata as a target language representation to analyze Java. Methods from computational linguistics were also applied to generate good Finite State Automata approximation of CFGs [26]. This approach is not so efficient than the other string analyzes because the source of data was not tracked and it must also determine Finite State Automata between each operations, so not a practical approach to find XSS vulnerabilities [44]. The same approach has been followed by Minamide to design a string analysis for PHP which is not approximating CFGs to Finite State Automata. This technique checks the presence of "<script>" tag in the whole document.

As web applications more often have their own scripts and also there are several other ways to invoke a JavaScript interpreter the approach is not at all practical to find XSS vulnerabilities.

Software Testing Approaches:

A number of black-box testing, fault injection and behavior monitoring to web applications approaches has been used by Y. Huang, S. Huang, Lin, and Tsai in order to predict the presence of vulnerabilities [44]. This approach combines user experience modeling as black-box testing and user-behavior simulation [40]. There are many other projects where a similar kind of approach has been followed like APPScan [46], WebInspect[45], and ScanDo [47]. As all these approaches were used to detect errors in the development life cycle, they might not be able to provide instant web application protection [26] and they cannot guarantee the detection of all flaws as well [48].

Taint Propagation Approach:

This kind of analysis is being used by many dynamic and static approaches, they use data flow analysis to track movement of information flow from origin to end. [18, 26, 27, 28, 29] There are some assumptions in this approach: The application is secure if a sanitization operation is performed from start to end in the in all the paths. [30]. It is not considered a good idea to have faith on user's filter and not to check the sanitization function because there are some Cross-Site Scripting attack vectors which can easily bypass many filters considered to be strong. Hence, a strong security mechanism is not provided here. [36].

Using Untrusted Scripts:
This is a method in which a list of untrusted scripts obtained from user given data is used to detect harmful scripts. Wassermann and Su's recent approach in [41] is an example of this technique. The method followed here was building policies and generating regular expressions of untrusted tags and then checking if there is an intersection between CFG, generated from String taint static analysis and regular expression generated. If the result was positive then further actions had to be taken. It is believed that untrusted list use is easy and not a good idea. The same has been stated in the document of OWASP. [4] It is clearly mentioned in the document not to use "blacklist" validation to detect XSS in input or to encode output. Search and replacement of a few characters ("<", ">") which are considered as bad is weak and has been attacked successfully. There are a number of different kinds of XSS which can easily bypass blacklist validation."

4.2.2 Dynamic Analysis Approach

Syntactical Structure Approach:
Su, and Wassermann in [36] suggested an approach which states that when there is a successful injection attack there is a change in the syntactical structure of the exploited entity. So, they have presented an approach where syntactic structure of output string is checked to detect malicious payload. For tracking sub-string from source to sinks they increased the user input with metadata. The modified parser was helped by this metadata to check the syntactical structure of the dynamically generated string by indicating start and end point of the user given input. Moreover the process was blocked if there was a sign of any abnormality. This approach was found to be quite successful while it detects any injection vulnerabilities other than XSS. Hence, it is not sufficient to avoid this sort of workflow vulnerabilities which are result of interaction between multiple modules [49].

Interpreter-based Approaches:
This approach has been suggested by Pietraszek, and Berghe in which there is use of instrumenting interpreter to track untrusted data at the character level and for identifying vulnerabilities that use context-sensitive string evaluation at each susceptible sink [32]. This technique is good and also able to detect vulnerabilities as security assurance is added by modifying the interpreter, however this approach of modifying interpreter is not easily feasible to some other famous and widely used web programming languages, such as Java, Jsp, Servlets [36].

Browser-Enforced Embedded Policies Approach:
The browser is provided with a white list of all the benign scripts to protect user it from malicious code [25]. This was a good idea which allows only the scripts in the provided list to run; however, since there is a lot of difference in the parsing mechanism of different browsers a successful filtering system of one browser may not be successful for other. Hence, although the technique explained in this paper is quite successful against these kinds of situations but a modification is required to be done in all the browsers to enforce the policy. So, the problem of scalability comes into existence from the point of view of web applications [39]. Furthermore every client needs to have this modified version of browser on their system.

Proxy-based Approach:
A web proxy could be used to prevent transferring of any sensitive information from a victim's site to any other site. Ex. - Noxes [20]. Malware is detected and blocked by this application-level firewall. There is a fine grained control provided to users on every connection coming or leaving the local machine. Firewall always prompts the user in case there is any mismatch between the connections coming or leaving the local machine and the rules set for it. Now user is responsible to decide whether to allow or block these connections. Similar kind of approaches has been followed in [43], [19], and [48]. It is not sufficient to Blacklist a link to prevent cross-site Scripting attacks. Huang et al. suggested, proxy-based approach does not provide a method to find errors, moreover it requires a watchful configuration [26]. This approach could definitely increase false positive as they protect the unpredictable link with no examination of the fault [40].

4.2.3 Static and Dynamic Analysis Approach

Lattice-based Approach:
There is a tool called WebSSARI which combines static and runtime features and find security vulnerabilities by applying static taint propagation analysis [26]. WebSSARI follows typestate and lattice model and uses flow sensitive, intra-procedural approach to determine vulnerability. When this tool knows that tainted data has reached sensitive function, it automatically puts runtime guards which are also called as sanitization routines [49]. There is a big drawback with this technique that it gives a large number of false negative and positive because of its intra-procedural type-based analysis [18]. Furthermore this approach also takes results from users' designed filters as safe. Hence, the real vulnerabilities might be missed, as it is quite possible that malicious payload may not be detected by designated filtering function.

4.3 Evaluation of XSS Approaches

Ten methodologies to detect XSS described in this paper has been evaluated in Table 6. The first column contains the approaches for different tools suggested. Low status indicates that the tool stated in this approach is unable to solve the problem. If it has a High status then it means that the tool stated in this approach is able to resolve the problem successfully and if it is Medium, the approach may be able to solve some part of the issue.

Table 7 figures out the false positive rate of those tools on the basis of the results published in different papers. Some of the results contain "NA" that means, there is not enough information available to summarize them. In Table 7 High states says that they generate more false positive which is a huge disadvantage of any tool.

5. Conclusion

There are many approaches and frameworks implemented in different web applications, security is still one of the major issues all across the globe. In this paper we have gone through specific cases of SQL injection and XSS attacks to web applications. We found out how existence of XSS and SQL injection vulnerabilities in web applications has huge

Table 6: Existing Methods' Capability to Resolve Problems

Approach\Problems	Browser-specific	DOM-based	Static-Script	Multi-Module
Browser-Enforced Embedded Policies	High	High	High	Low
Bounded Model Checking	Low	Low	Low	Low
Interpreter-based	High	Low	High	Low
Lattice-based Analysis	Low	Low	Low	Low
Preventing XSS Using Untrusted Scripts	Medium	Low	Low	Low
Proxy-based Solutions	Low	Low	Low	Low
Software Testing Techniques	Low	Low	Low	Low
String Analysis	Low	Low	Low	Low
Syntactical Structure Analysis	Low	Low	Low	Low
Taint Propagation Analysis	High	High	High	High

Table 7: False Positive Rate of Existing Methods

Approaches	False Positive
Browser-Enforced Embedded Policies	Low
Bounded Model Checking	NA
Interpreter-based	Medium
Lattice-based Analysis	High
Preventing XSS Using Untrusted Scripts	Medium
Proxy-based Solutions	Medium
Software Testing Techniques	NA
String Analysis	Medium
Syntactical Structure Analysis	Low
Taint Propagation Analysis	High

risk not only for the application but also for users as well. We surveyed various existing approaches to detect and prevent these vulnerabilities in an application, giving a brief note on their advantages and disadvantages. All the approaches followed by different authors' leads to a very interesting solution; however some failures are associated with almost each one of them at some point. Furthermore some of them even do not provide reasonable security and can be easily bypassed by attackers. Also a few of them are so complex it is almost impractical for a user to use in real situations. The key finding of this paper is analytical survey report on various types of SQL injection and XSS attacks against different methods defined by several authors. All the methods have been assessed on the basis of their performance and feasibility.

6. Acknowledgement

Sincere thanks to Konstantin Beznosov, San-Tsai Sun and anonymous reviewers for their valuable comments and help in early drafts of this paper.

7. References

[1] Sruthi Bandhakavi, Prithvi Bisht, P. Madhusudan, CANDID: Preventing SQL Injection Attacks using Dynamic Candidate Evaluations, 2007, Alexandria, Virginia, USA, *ACM*.

[2] A Tajpour, A., Masrom, M., Heydari, M.Z., and Ibrahim, S., SQL injection detection and prevention tools assessment. *Proc. 3rd IEEE International Conference on Computer Science and Information Technology* (ICCSIT'10) 9-11 July (2010), 518-522

[3] L Halfond W. G., Viegas, J., and Orso, A., A Classification of SQL-Injection Attacks and Countermeasures. *In Proc. of the Intl. Symposium on Secure Software Engineering*, Mar. (2006).

[4] http://www.owasp.org/index.php/Top_10_2010-A1-Injection, retrieved on 13/01/2010

[5] Xin Jin, Sylvia Losborn. Architecture for Data Collection in Database Intrusion Detection System.

Secure Data Management. Pages 96-107.*Springer Berlin* /Heidelberg. 2007.

[6] Shin, Y., Williams, L., and Xie, T., *SQLUnitGen: Test Case Generation for SQL Injection Detection*. North Carolina State University, Raleigh Technical report, NCSU CSC TR 2006-21 (2006).

[7] T Haixia, Y. and Zhihong, N., A database security testing scheme of web application. *Proc. of 4th International Conference on Computer Science & Education 2009* (ICCSE '09), 25-28 July (2009), 953-955

[8] Fu, X., Lu, X., Peltsverger, B., Chen, S., Qian, K., and Tao, L., A Static Analysis Framework for Detecting SQL Injection Vulnerabilities. *Proc. 31st Annual International Computer Software and Applications Conference 2007* (COMPSAC 2007), 24-27 July (2007), 87-96.

[9] Ruse, M., Sarkar, T., and Basu. S., Analysis & Detection of SQL Injection Vulnerabilities via Automatic Test Case Generation of Programs. *Proc. 10th Annual International Symposium on Applications and the Internet* (2010), 31-37

[10] Thomas, S., Williams, L., and Xie, T., On automated prepared statement generation to remove SQL injection vulnerabilities. *Information and Software Technology*, Volume 51 Issue 3, March (2009), 589–598 1

[11] Roichman, A., Gudes, E., Fine-grained Access Control to Web Databases. *Proceedings of 12th SACMAT Symposium, France* (2007).

[12] Kemalis, K. and T. Tzouramanis. SQL-IDS: A Specification-based Approach for SQLinjection Detection. SAC'08. Fortaleza, Ceará, Brazil, *ACM* (2008), 2153 2158.

[13] Junjin, M., An Approach for SQL Injection Vulnerability Detection. *Proc. of the 6th International Conference on Information Technology: New Generations*, Las Vegas, Nevada, April (2009), 1411-1414.

[14] TBoyd S.W. and Keromytis, A.D., SQLrand: Preventing SQL Injection Attacks. *Proceedings of the 2nd Applied Cryptography and Network Security (ACNS'04) Conference*, June (2004), 292–302. 8

[15] McClure, R.A. and Kruger, I.H., SQL DOM: compile time checking of dynamic SQL statements. *27th International Conference on Software Engineering* (ICSE 2005), 15-21 May (2005), 88- 96.

[16] http://www.codesmithtools.com.

[17] J. Garcia-Alfaro and G. Navarro-Arribas, "Prevention of cross-site scripting attacks on current web applications," in *Proceedings of the 2007 OTM confederated international conference on On the move to meaningful internet systems: CoopIS, DOA, ODBASE, GADA, and IS* - Volume Part II, ser. OTM'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 1770–1784

[18] Y. Xie, and A. Aiken, "Static detection of security vulnerabilities in scripting languages," *In Proceeding of the 15thUSENIX Security Symposium*, July 2006, pp. 179-192.

[19] "AppShield," Sanctum Inc. http://sanctuminc.com, 2005.

[20] E. Kirda, C. Kruegel, G. Vigna, and N. Jovanovic, "Noxes: A client-side solution for mitigating cross site scripting attacks," *In Proceedings of the 21st ACM symposium on Applied computing, ACM*, 2006, pp. 330-337.

[21] Microsoft. HotMail: The World's FREE Web-based E-mail. http://hotmail.com/

[22] .Jagatic, T., Johnson, N., Jakobsson, M., and Menczer, F. Social Phishing. To appear in Communications of the ACM.

[23] Wikipedia, http://wikipedia.org.

[24] Grossman, RSNAKE, PDP, Rager, and Fogie, "XSS Attacks: Cross-site Scripting Exploits and Defense," *Syngress Publishing Inc*, 2007.

[25] T. Jim, N. Swamy, and M. Hicks, "BEEP: Browser-Enforced Embedded Policies," *In Proceedings of the 16th International World Wide Web Conference, ACM*, 2007, pp. 601-610.

[26] Y.-W. Huang, F. Yu, C. Hang, C. H. Tsai, D. Lee, and S.Y. Kuo,"Securing web application code by static analysis and runtime protection," *In Proceedings of the 13 th International World Wide Web Conference*, 2004.

[27] V.B. Livshits, and M.S. Lam, "Finding security errors in Java programs with static analysis," *In proceedings of the 14th Usenix security symposium*, August 2005, pp. 271-286.

[28] "JavaScript Security: Same origin," Mozilla Foundation, http://www.mozilla.org/projects/security/components/same-origin.html, February 2006

[29] N. Jovanovic, C. Kruegel, and E. Kirda, "Precise alias analysis for syntactic detection of web application vulnerabilities," *In ACM SIGPLAN Workshop on Programming Languages and Analysis for security*, Ottowa, Canada: June 2006.

[30] D. Balzarotti, M. Cova, V. Felmetsger, N.Jovanovic, E. Kirda, C. Kruegel, and G. Vigna, "Saner: Composing Static and Dynamic Analysis to Validate Sanitization in Web Applications," *In IEEE symposium on Security and Privacy*, 2008.

[31] Amirtahmasebi, K., Jalalinia, S.R., and Khadem, S., A survey of SQL injection defense mechanisms. *International Conference for Internet Technology and Secured Transactions (*ICITST 2009), 9-12 Nov. (2009), 1-8

[32] T. Pietraszek, and C. V. Berghe, "Defending against Injection Attacks through Context-Sensitive String Evaluation," *In Proceeding of the 8th International Symposium on Recent Advance in Intrusion Detection* (RAID), September 2005.

[33] Buehrer, G., Weide, B.W., and Sivilotti, P.A.G., Using Parse Tree Validation to Prevent SQL Injection Attacks. *Proc. of 5th International Workshop on Software Engineering and Middleware, Lisbon, Portugal* (2005) 106–113.

[34] Bisht, P., Madhusudan, P., and Venkatakrishnan, V.N., CANDID: Dynamic Candidate Evaluations for

Automatic Prevention of SQL Injection Attacks. *ACM Transactions on Information and System Security, Volume 13 Issue 2*, (2010), doi>10.1145/1698750.1698754.

[35] Ali, S., Shahzad, S.K., and Javed, H., SQLIPA: An Authentication Mechanism Against SQL Injection. *European Journal of Scientific Research, Vol*. 38, No. 4 (2009), 604-611.

[36] Z. Su and G. Wassermann, "The essence of command Injection Attacks in Web Applications," *In Proceeding of the 33rd Annual Symposium on Principles of Programming Languages, USA: ACM, January* 2006, pp. 372-382.

[37] Roichman, A., and Gudes, E., DIWeDa - Detecting Intrusions in Web Databases. Atluri,V. (ed.) DAS 2008. LNCS, vol. 5094, Springer, Heidelberg (2008), 313-329.

[38] Baker, R.A., Code Reviews Enhance Software Quality. *In Proceedings of the 19th international conference on Software engineering (ICSE'97), Boston, MA*, USA (1997), 570-571.

[39] P. Bisht, and V.N. Venkatakrishnan, "XSS-GUARD: Precise dynamic prevention of Cross-Site Scripting Attacks," *In Proceeding of 5th Conference on Detection of Intrusions and Malware & Vulnerability Assessment, LNCS* 5137, 2008, pp. 23-43.

[40] Y.-W Huang, F. Yu, C. Hang, C. –H. Tsai, D. Lee, and S. –Y. Kuo. "Verifying Web Application using BoundedModel Checking," *In Proceedings of the International Conference on Dependable Systems and Networks*, 2004.

[41] G. Wassermann, and Z. Su, "Static detection of cross-site Scripting vulnerabilities," *In Proceeding of the 30th International Conference on Software Engineering*, May 2008.

[42] A.S. Christensen, A. Móller, and M.I. Schwartzbach, "Precise analysis of string expression," *In proceedings of the 10th international static analysis symposium, vol. 2694 of LNCS*, Springer-Verlag, pp. 1-18.

[43] N. Jovanovic, C. Kruegel, and E. Kirda, "Pixy: A static analysis tool for detecting web application vulnerabilities (short paper)," *In 2006 IEEE Symposium on Security and Privacy*, Oakland, CA: May 2006.

[44] Y.-W. Huang, S.-K. Huang, T.-P. Lin, and C.-H. Tsai, "Web application security assessment by fault injection and Behavior Monitoring," *In Proceeding of the 12th international conference in World Wide Web, ACM, New York, NY*, USA: 2003, pp.148-159.

[45] "Web Application Security Assessment," SPI Dynamics Whitepaper, SPI Dynamics, 2003.

[46] "Web Application Security Testing – AppScan 3.5," Sanctum Inc., http://www.sanctuminc.com.

[47] "InterDo Version 3.0," Kavado Whitepaper, Kavado Inc. , 2003

[48] D. Scott, and R. Sharp, "Abstracting Application-Level Web Security," *In Proceeding 11thinternational World Wide Web Conference, Honolulu, Hawaii*: 2002, pp. 396-407

[49] D. Balzarotti, M. Cova, V. V. Felmetsger, and G. Vigna, "Multi-Module Vulnerability Analysis of Web-based Applications," *In proceeding of 14th ACM Conference on Computer and Communications Security*,Alexandria, Virginia, USA: October 2007.