Web Application Vulnerabilities: A Survey

Vandana Dwivedi
PG Scholar, CSE.
RITS, Bhopal (India)

Himanshu Yadav
Asst. Prof., CSE dept.
RITS, Bhopal (India)

Anurag Jain
HOD (CS)
RITS, Bhopal (India)

ABSTRACT
In the last few years, the discovery of World Wide Web (WWW) has grown very much. Today, WWW applications are routinely utilized in security critical environments, like e-commerce, medical, financial, and military systems etc. WWW systems are an organization of infrastructure elements, like web databases and servers, and application-specific code, such as HTML scripts and CGI programs etc. While the core elements are usually developed by knowledgeable programmers with valid security skills this ensuing vulnerable web-based applications and accessible to the complete web, creating easily-abusing access points for the conciliation of entire networks. During this paper, we survey the current approaches to internet vulnerability analysis and that we propose a classification along two characterizing: detection and prevention model and study these methods. Furthermore we describe the foremost regular attacks in contrast to web-based applications and explore the effectiveness of sure analysis techniques in characteristic specific categories of flaws.

Keywords
Web applications security, SQL injection, Cross-side scripting, Cross-site request forgery, vulnerabilities

1. INTRODUCTION
In the recent years, the globe Wide internet (WWW) has witnessed a staggering growth of the many on-line internet applications that are developed for meeting numerous purposes. Now-a-days, nearly everybody connected with 'computer technology' is somehow connected on-line. To serve this immense variety of users, nice volumes of knowledge are stored in internet application databases in several components. Timely, the consumer should move with the backend databases via the user interfaces for various tasks such as: modification information, making queries, extracting information, etc. For these operations, vogue interface plays an important role, the quality of that features a pleasant impact on the protection of the keep information inside the knowledge. An unsecure web application might enable crafted injection and malicious update on the backend information. This trend will cause numerous damages and thefts of trustworthy users' sensitive knowledge by unauthorized users. Within the worst case, the assaulter might gain full management over the net application and entirely destroy or damage the system.

SQL Injection may be a kind of injection or attack in an exceedingly internet application, during which the aggressor provides structured question Language (SQL) code to a user input box of an online type to gain unauthorized and unlimited access. The attacker's input is transmitted into associate degree SQL question in such the way that it forms an SQL code [1], [2]. In fact, SQL Injection is classified as the top-10 2010 net application vulnerabilities experienced by net applications consistent with OWASP (Open net Application Security Project) [3].

SQL Injection Vulnerabilities (SQLIV's) unlocks entrance for hackers to explore and attack. Hence, they show a severe hazard for net application components. Main concept of SQLIVs is sort of straightforward and well known insufficient validation of user input [1].

To overcome from such kind of vulnerabilities, several techniques are suggested like manual approach, machine-controlled approach; secure secret writing practices, static analysis, exploitation ready statements, and then forth. Though, planned approaches have achieved their goals to some extent, SQL Injection Vulnerabilities in net applications stay as a significant concern among application developers.

Web applications
The business logic of an internet application is enforced at an internet server and a backend server, and publicised by a uniform resource locator (URL). The internet server is understood by its name. The most infrastructure part on the consumer aspect is that the browser, that has no name apart from the client's IP address. Browser and server communicate via a transport protocol. A transport protocol defines data formats, additionally conjointly algorithms for packaging and unpacking application payloads. Fig. 1 shows the fundamental architecture of information flow in a web application. The transport protocol is HTTP; the info format is hypertext mark-up language (HTML) and Cascading style Sheets (CSS). The user calls AN application by clicking on its URL. The client's browser then sends a communications protocol request to the net server. A script at the net server extracts input from the consumer knowledge and constructs a request to a backend application server, e.g. AN SQL query to a database. The web server receives the result from the backend server and returns a hypertext mark-up language (HTML) result page to the consumer. The client's browser displays the result page. To show a page, the browser creates an interior representation for it.

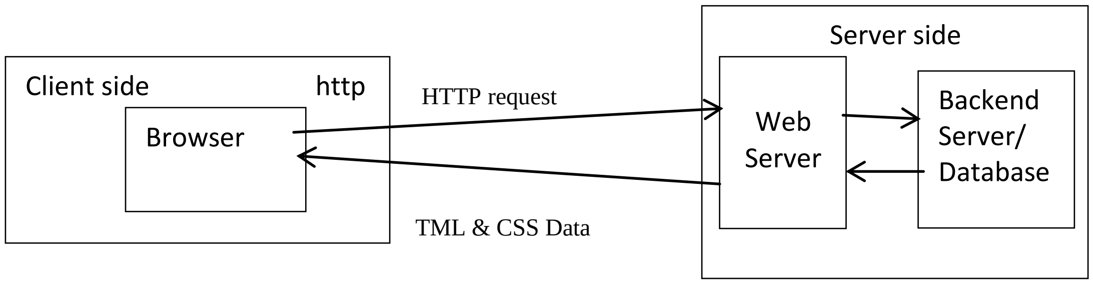

Fig. 1: Architecture of Web Application

This representation is that the supposed Domain Object Model (DOM) [4]. Once the browser receives a hypertext mark-up language (HTML) page it parses the hypertext mark-up language into the document. Body of the DOM. Objects like document, URL, document location, and document referrer get their values consistent with the browser's read of the present page.

After this primary information, the remains of the paper are structured as follows: Section 2, describe types of the vulnerabilities and attacks in web application. Section 3 presents classification foremost common SQL Injection attacks. Section 4 notes down literature survey related to SQLIA, Section 5 notes down the accessible countermeasures to tackle varied SQL Injection attacks and a comparative analysis of varied attacks and schemes, section 6 states detailed research scope and at last, Section 7 concludes the paper noting the contribution of this work aboard mentioning future analysis objectives.

2. VULNERABILITIES IN WEB APPLICATION

In general, there are three kinds of security vulnerabilities among web applications at completely different levels: (1) input validation vulnerability at the single request level, (2) session management vulnerability at the session level, and (3) application logic vulnerability at the extent of the whole application. In what follows, description of the above three kinds of vulnerabilities are presented and the common attacks that exploit these vulnerabilities.

2.1 Input Validation Vulnerabilities

A common security observes is input data validation, since user input data can't be trusty. Data validation is that the method of guaranteeing that a program operates on clean, correct and helpful input data. Once inputs don't seem to be sufficiently or properly valid, attackers are ready to craft distorted inputs, which might alter program executions and gain unauthorized access to resources. Input validation vulnerability may be a durable drawback in software system security. Incorrect or depleted input validation may invite a range of attacks, like buffer overflow attacks and code injection attacks. Web applications might contain a large vary of input validation vulnerabilities. Since the whole web request, as well as request headers and payload data, is beneath the entire management of users, a web application must make sure that user inputs are processed and utilized in a very secure manner throughout the execution.

SQLI (SQL Injection):- SQL Injection is a code injection technique where attacker injects malicious code in to strings that are later passed to SQL server for execution. A web application is at risk of SQL injection attacks once malicious content will flow into SQL queries while not being absolutely sanitized, that permits the offender to trigger malicious SQL operations by injecting SQL keywords or operators. For example, the offender will append a separate SQL query to the present query, causing the application to drop the complete table or manipulate the comeback result. Malicious SQL statements may be introduced into a vulnerable application victimization many various input mechanisms [1] as well as user inputs, cookies and server variables

Cross-site Scripting (XSS): vulnerabilities arise from associate application's failure to properly validate user input before it's came to a user. Mistreatment this vulnerability, associate offender will force a consumer, like a user application, to execute attacker-supplied code, like JavaScript, within the context of a trusty computing machine [5]. As a result, the attacker's code is granted access to security-critical data that was issued by (or is associated with) the trusty website.

2.2 Session Management Vulnerabilities

Session management is essential for a web application to keep track of user inputs and maintain application states. Within the OWASP top-ten security risks [3], three are related to session management vulnerabilities: (1) Broken Authentication and Session Management, (2) Cross-Site Request Forgery and (3) Insufficient Transport Layer Protection.

In web application development, session management is accomplished through the collaboration between the client and the server. A common approach is that the server sends the client a unique identifier (i.e., a session ID) upon successful user authentication, through which the server recognizes the client on subsequent requests and indexes his session variables stored at the server side. Since session ID is the only proof of the client's identity, its confidentiality, integrity and authenticity need to be ensured to avoid session hijacking.

First, the session ID should be random for each client's visit and expire after a short period of inactivity. Weak session identifier generation allows attackers to hijack the victim's web sessions by predicting his session ID. Second, transmission of the session ID should always be protected by a secure transport layer protocol (i.e., over SSL). Otherwise, attackers are able to sniff the session ID and hijack the session. Third, the client needs to make sure that his session ID is provided by the server and is unique. Adopting a session ID from an external source opens up a vulnerability to session fixation, where attackers can set the session ID to a value that is known to them.

Securing the session ID alone is not sufficient for secure session management. Session hijacking can also be achieved through malicious web requests that are associated with a valid session ID. Cross-site request forgery (CSRF) is a popular attack of this type, where attackers trick the victim into sending crafted web requests on their behalf. The vulnerable web application cannot differentiate if the incoming web requests are malicious, since they are associated with valid session information. For example, attackers may forge a web request that instructs a vulnerable banking website to transfer the victim's money to his account. Login CSRF [6], on the other hand, tricks the victim into logging in to a target website using the attacker's credential through a forged request. This attack allows the attacker to harvest the information about the victim's activities under the attacker's account.

2.3 Application Logic Vulnerabilities

The decentralized structure of web applications poses significant challenges to the implementation of business logic. First, since web application modules can be accessed directly through their URLs, *interface hiding* mechanism has been commonly used as a measure for access control in web applications. However, this mechanism alone, which follows then principle of "security by obscurity", is not sufficient to enforce the control flow of a web application. Application logic vulnerabilities are highly dependent on the intended functionality of a web application. For example, a vulnerable e-commerce website may have a specific logic vulnerability that allows attackers to apply the same coupon multiple times to reduce prices. Despite the heterogeneous application functionalities, there are several types of logic flaws that correspond to common business logic patterns in many applications.

One common type is access control vulnerability, which allows attackers to access unauthorized sensitive information or operations. Another type is workflow violation, which allows attackers to violate the intended steps within business workflows. For example, a vulnerable e-commerce website may allow attackers to bypass the tax calculation step during the checkout procedure.

The class of attacks that target application logic vulnerabilities are generally referred to as logic attacks or state violation attacks Depending on how attacks are launched, they can be given several other terms. Forceful browsing [7] is one attack vector, where attackers directly point to hidden but predictable web links to access sensitive information. Parameter tampering [8] is launched by manipulating certain values in web requests to exploit application logic.

3. TYPES OF SQL INJECTION ATTACKS

Among numerous forms of SQLI attacks, some are often utilized by the attackers. It's imperative to understand the normally used major attacks among all out there attacks. Hence, during this section, an in-depth explanation is presented to investigate a number of the foremost common SQL Injection attacks.

3.1 Tautology

SQL injection codes are injected into one or more conditional statements so that they are always evaluated to be true.This type of attack injects SQL tokens to the conditional query statement to be evaluated always true.

*SELECT*FROM member WHERE member_username =' ' OR 1=1 – AND member_password =' '*

In the above query, 1=1 always true and if the application not validates the user input properly then all the records from the database will be fetched to the application. Under this technique, the following types and scenarios of attacks may be occurs:

- String SQL Injection
- Comments Attack
- Numeric SQL Injection

3.2 Illegal/Logically Incorrect Queries

The purpose of this attack is to understand the database properties. When this type of query is executed on the database it displays an error messages. By proper understanding and analyzing of this error the attacker will identify the backend DBMS details. Using error messages rejected by the database to find useful data facilitating injection of the backend database

SELECT name FROM account WHERE member_password='1\'

3.2 Union Query

Injected query is joined with a safe query using the keyword UNION in order to get information related to other tables from the application.This attack uses the union operator which performs unions between two or more queries.

*SELECT * FROM members WHERE member_username='user123' UNION SELECT*FROM member WHERE member_username='admin'—AND member_password=' '*

3.3 Piggy-Backed Queries

In this type of attacks, attacker appends an extra query to the original query.

3.4 Stored Procedures

Many databases have built-in stored procedures. The attacker executes these built-in functions using malicious SQL Injection codes. A stored procedure is a group of Transact-SQL statements compiled into a single execution plan. Depend on specific stored procedure on the database there are different kinds of attack. A stored procedure example is given in the following.

CREATE PROCEDURE

authenticateUser (IN username VARCHAR (16), IN password VARCHAR (32))

BEGIN

*SELECT * FROM members WHERE member_username = username AND member_password = password;*

END

Above stored procedure is also vulnerable to both the tautologies and piggybacked queries.

3.5 Inference

In this type of attack, the attacker observes the behaviour of web application based on a series of true/false questions and timing delays. By careful observing the behaviour of application the attacker identifies the vulnerable parameters in the application. These attacks are composed of two types: blind Injection and timing attacks, in the former one the attacker issues true/false type of questions to the database and latter one attacker gather information from a database by observing in the timing-delays in the database responses.

3.6 Alternate Encodings

It targets to avoid being known by secure defensive coding and automatic prevention mechanisms. It's sometimes combined with different attack techniques. During this technique, attackers modify the injection query by victimization alternate encoding, like hexadecimal, ASCII, and Unicode. As a result they will throw off developer's filter that scans input queries for special known "bad character".

4. LITERATURE SURVEY

Many techniques have been used or suggested to detecting and preventing SQL Injection Vulnerabilities in Web applications. Here, explanation of the prominent solutions and their working methods is presented in brief to let the readers know about the core ideas behind each work.

A tool named WebSSARI [9], revealed in 2004, and is one in every of the primary works that applies static taint propagation analysis to finding security vulnerabilities in PHP applications. WebSSARI targets three specific varieties of vulnerabilities: cross-site scripting, SQL injection, and general script injection. The tool uses flow-sensitive, intra-procedural analysis supported a lattice model and sort state. Above all, the PHP language is extended with 2 type-qualifiers, specifically tainted and unblemished, and therefore the tool keeps track of the type-state of variables. The tool uses 3 user-provided files, known as prelude files: a file with preconditions to any or all sensitive functions (i.e., the sinks), a file with post conditions for acknowledged cleansing functions, and a file specifying all attainable sources of entrusted input. So as to undamaged the contaminated information, the information needs to be processed by a cleansing routine or to be forged to a secure kind. Once the tool determines that tainted information reaches sensitive functions, it mechanically inserts runtime guards, that area unit cleansing routines.

Another approach together supported static taint propagation analysis, to the detection of input validation vulnerabilities in PHP applications is described in [10, 11]. A flow sensitive, inter-procedural and context-sensitive info flow analysis is used to identify intra-module XSS and SQL injection vulnerabilities. The approach is enforced during a very tool, referred to as Pixy that's that the foremost complete static PHP analyzer in terms of the PHP choices shapely. To the only information, it is the sole publicly-available tool for the analysis of PHP-based applications.

The work by [12], describes a three-level approach to look out SQL injection vulnerabilities in PHP applications. First, symbolic execution is used to model the impact of statements among the basic blocks of intra-procedural management Flow Graphs (IFGs). Then, the following block define is used for intra procedural analysis, where a typical reach ability analysis is used to induce perform define. In conjunction with different information, each block define contains a bunch of locations that were undamaged among the given block. The block summaries area unit composed to return up with perform define, that contains the pre- and post-conditions of perform. The preconditions for perform contain a derived set of memory locations that ought to be compelled to be alter before the perform invocation, whereas the post conditions contain the set of parameters and international variables that area unit alter among perform. To model the results of cleansing routines, the approach uses a programmer-provided set of possible cleansing routines, considers certain forms of casting as a cleansing technique, and, in addition, it keeps info of sanitizing regular expressions, whose effects area unit specific by the individual. Once perform summaries area unit computed, they are utilized in inter-procedural analysis to seem for possible SQL injections.

The work by [13] is another example of an approach that uses a model of “normality” to find injection attacks, like XSS, XPath injection, and shell injection attacks. However, this implementation, known as SqlCheck is meant to find SQL injection attacks solely. The approach works by trailing substrings from user input through the program execution. The trailing is enforced by augmenting the string with special characters, which mark the beginning and therefore the finish of every substring. Then, dynamically-generated queries area unit intercepted and checked by a changed SQL programmed. Mistreatment the meta-information provided by the substring markers, the program is ready to work out if the question syntax is changed by the substring derived from user input, and, therein case, it blocks the question.

Another example is the work by [14] that takes a look at a new and unexplored class of vulnerabilities in the domain of web applications. In particular, the paper looks at race condition vulnerabilities that can arise in web applications interacting with a back-end database. A race condition may occur in a multi-threaded environment between two database queries if data accessed by one query can be modified by another one. In a multi-threaded application, the shared data in the database might not be consistent between the two queries if code that was designed to be executed sequentially is executed concurrently. The authors propose a dynamic approach to identify this class of vulnerabilities, in which all database queries generated by a running program are logged and analyzed (offline) for data dependencies.

A good example of an approach based on a model of expected behavior is the work of [15], whose tool is called AMNESIA [15]. AMNESIA is particularly concerned with detecting and preventing SQL injection attacks for Java-based applications. During the static analysis part, the tool builds a conservative model of expected SQL queries. Then, at run-time, dynamically-generated queries are checked against the derived model to identify instances that violate the intended structure of a query. AMNESIA uses Java String Analysis (JSA) [16], a static analysis technique, to build an automata-based model of the set of legitimate strings that a program can produce at given points in the code. AMNESIA also leverages the approach proposed by Gould, Su, and Devanbu [17] to statically check type correctness of dynamically-generated SQL queries.

More precisely, author defines a SQL injection as the attack in which the logic or semantics of a legitimate SQL statement is changed due to malicious injection of new SQL keywords or operators. Thus, to detect such attacks, the semantics of dynamically-generated queries must be checked against a derived model that rep-resents the intended semantics of the query.

SQL DOM [18] utilizes database queries encapsulation for authentic access to databases. They use a type-checked API which cause query building process is systematic. Therefore by API they implement coding finest practices for instance input filtering and strict user input type checking. The drawback of the approaches is that developer should be trained new programming standard or query-development practice.

Another approach in this category is SQL-IDS [19] which focus on writing specifications for the web application that describe the intended structure of SQL statements that are produced by the application, and in automatically monitoring the execution of these SQL statements for violations with respect to these specifications.

SQLPrevent [20] is consists of AN HTTP request interceptor. The first data flow is changed once SQLPrevent is deployed into an online server. The HTTP requests are saved into this thread-local storage. Then, SQL interceptor intercepts the SQL statements that are created by internet application and pass them to the SQLIA detector module. Consequently, HTTP request from threads native storage is fetched and examined to work out whether or not it contains an SQLIA. These malicious SQL statements would be prohibited to be sent to information, if it's suspicious to SQLIA.

Swaddler [21] is innovative scheme to the anomaly-based detection of attacks challenging web applications. Swaddler inspects the inside state of a web application and examines the interaction between the application’s crucial execution points and the application’s internal state. By doing this, Swaddler is capable to recognize attacks that try to fetch an application in an inconsistent, abnormal state, such as violations of the intended workflow of a web application.

In [22] author proposed the initial explanation of command injection attacks in the perception of web applications, and dispenses absolute algorithm for preventing them founded on context-free grammars and compiler parsing techniques. Author’s assessment is that, for an attack to be successful, the input that gets circulated into the database query or the output document must modify the intended syntactic organization of the query or document. This description and algorithm are common and concern to many forms of command injection attacks. This scheme is authenticate with SQLCHECK, an implementation for the setting of SQL command injection attacks. They assessed SQLCHECK on routine web applications with methodically compiled daily attack data as input. SQLCHECK produced no false positives or false negatives, incurred low runtime overhead, and applied straightforwardly to web applications written in different languages.

An attacker who knows nothing about the key to the randomization algorithm will inject code that is not valid for that randomized processor, reasoning a runtime exception. In work [23] utilizes the similar method to the difficulty of SQL injection attacks: they produce randomized instances of the SQL query language, by randomizing the template query within the CGI script and the database parser. To permit for easy retrofitting of their method to existing systems, they initiate a de-randomizing proxy, which alters randomized queries to appropriate SQL queries for the database.

5. CRITICAL ANALYSIS

Table 1 explains a digest of so far recognized countermeasures against SQL Injection. Now, let us see what these schemes are actually about. It would be tough to provide a transparent finding of fact that scheme or approach is that the best as each has some verified advantages for specific kinds of settings (i.e., systems).. Table 2 shows a chart of the schemes and their defense capabilities against varied SQLIAs. This table shows the comparative analysis of the SQL Injections bar techniques and also the attack sorts. Although several approaches are known as detection or prevention techniques, solely few of them were enforced in utility. Hence, this comparison isn't supported empirical expertise however rather it's an analytical analysis.

Table 1: Countermeasures of SQL Injection

Countermeasure	Description	Detection	Prevention
SQL-IDS [19]	A specification based approach to detect malicious intrusions	yes	Yes
AMNESIA [15]	This scheme identifies illegal queries before their execution. Dynamically-generated queries are compared with the statically-built model using a runtime monitoring	yes	Yes
SQLrand [23]	A strong random integer is inserted in the SQL keywords.	yes	Yes
SQL DOM [18	A set of classes that are strongly-typed to a database schema are used to generate SQL statements instead of string manipulation	yes	Yes
SQLGuard [24]	The parse trees of the SQL statement before and after user input are compared at a run time. The Web script has to be modified	yes	No
CANDID [25]	Programmer-intended query structures are guessed based upon evaluation runs over non-attacking candidate inputs	yes	No
SQLIPA [26]	Using user name and password hash values, to improve the security of the authentication process	yes	No
SQLCHECK [22]	A key is inserted at both beginning and end of user's input. Invalid Syntactic forms are the attacks. The key strength is a major issue	yes	No

Table 2: Various methods of different SQL Injection Attacks

Methods	Tautolog y	Logically Incorrect Queries	Union Query	Stored Procedure	Piggy-Backed Queries	Inference	Alternate Encodings
SQL-IDS [19]	yes	Yes	Yes	yes	yes	yes	Yes
AMNESIA [15]	yes	Yes	Yes	no	yes	yes	Yes
SQLrand [23]	yes	No	Yes	no	yes	yes	No
SQL DOM	yes	Yes	Yes	no	yes	yes	Yes
SQLGuard [17]	yes	Yes	Yes	no	yes	yes	Yes
CANDID [25]	yes	No	No	no	no	no	No
SQLIPA [26]	yes	No	No	no	no	no	No

SQLCHECK [22]	yes	Yes	Yes	no	yes	yes	Yes
WebSSARI	yes	Yes	Yes	yes	yes	yes	Yes

6. RESEARCH SCOPE

Although a considerable quantity of efforts are dedicated to addressing input validation vulnerabilities and attacks, many open problems are still not sufficiently addressed, and XSS remains the foremost common internet attack these days. Web application development usually give auto sanitization options a recent study [22] shows, they still cannot meet the entire necessities exhibit by trendy internet applications. Designing and reasoning context-sensitive sanitization routines still need substantial work. The identification of input validation vulnerabilities from legacy internet applications remains difficult.

Although taint-based techniques are incontestable to be very effective, they can't be directly applied to a large variety of recently developed internet applications. Web applications sometimes involve many technologies, languages, or elements that make it even tougher to trace user info flow and establish delicate second-order attacks. To address these problems, one single technique tends to be deficient. We've seen an increasing variety of works that combine two or additional techniques to attain higher performance, like hybrid taint analysis [21], string taint analysis [13]. The question of how to combine existing techniques during an inventive way to address the restrictions of single techniques is a stimulating analysis direction.

Even for the development of recent secure internet applications, it still needs consistent efforts from developers to follow secure coding practices to create strong session management mechanisms. Securing internet applications from logic flaws and attacks still remains an underexplored space. Solely a restricted variety of techniques are projected. Most of them solely address a particular form of application logic vulnerabilities, like authentication and access management vulnerabilities or inconsistencies between shopper and server validations [8][25][27]. The fundamental issue in Endeavour general logic flaws is the absence of application logic specification. The absence of a general and automatic mechanism for characterizing the application logic is one among the inherent reasons for the lack of most application scanners and firewalls to handle logic flaws and attacks [27].

Several recent works attempt to develop a general and systematic methodology for automatically inferring the specifications for internet applications that in turn facilitates automatic and sound verification of application logic. One among the key observations of those works [21] [28] is that the application's meant behavior is typically disclosed below its traditional execution, once users follow the navigation ways. In [30], similar assumption is formed for well-behaved clients, wherever they're expected by the server to invoke the URLs during a specific sequence with specific arguments. In order to infer the application logic, one category of strategies leverages the program source code [21] [29]. As a result, the inferred specification extremely depends on however the application is structured and enforced (e.g., the definition of a program operates or block). The accuracy of the inferred specification is additionally littered with its capability of handling language details. Another category of strategies infers the application specification by observant and characterizing the application's external behavior [28]. The noisy info discovered from the external behaviors might result in an inaccurate specification through these strategies.

7. CONCLUSION

In recent years, internet applications became hugely common, and these days they're habitually utilized in numerous security-critical environments. Because the use of internet applications for essential services has accumulated, the amount and class of attacks against these applications have full-grown moreover. So far, the analysis communities primarily targeted on effort vulnerabilities that result from insecure info flow in internet applications, like cross-site scripting and SQL injection. Whereas relative success was reached in characteristic appropriate techniques and approaches for managing this kind of vulnerabilities, very little has been explored regarding vulnerabilities that result from blemished application logic.

Though several approaches and frameworks are known and enforced in several interactive internet applications, security still remains a serious issue. SQL Injection prevails in concert of the top-10 vulnerabilities and threat to on-line businesses targeting the backend databases. During this paper, we've got reviewed the foremost common existing SQL Injections related problems. We tend to believe that the work would be helpful each for the overall readers of the subject as well as for the practitioners. As a future work, we might wish to develop a step which will efficiently tackle the innovative SQL Injection attacks and fix the maximum amount vulnerability as potential. Hackers are actually very innovative and because the time is passing by, new attacks are being launched that will want new ways that of considering the solutions we presently have.

8. REFERENCES

[1] Halfond, W. G., Jeremy Viegas, and Alessandro Orso. "A classification of SQL-injection attacks and countermeasures" In Proceedings of the IEEE International Symposium on Secure Software Engineering, Arlington, VA, USA, pp. 13-15. 2006.

[2] Tajpour, Atefeh, Maslin Masrom, Mohammad Zaman Heydari, and Suhaimi Ibrahim. "SQL injection detection and prevention tools assessment" In Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on, vol. 9, pp. 518-522. IEEE, 2010

[3] Top 10 2010-A1-Injection, available at: http://www.owasp.org/index.php/Top_10_2010-A1-Injection, last accessed 11 June, 2013.

[4] Le He´garet P, Whitmer R, Wood L. Document object model (DOM). W3C Recommendation, <http://www.w3.org/DOM/>; January 2005.

[5] A. Klein. "Cross Site Scripting Explained" Technical report, Sanctum Inc., June 2002.

[6] Gmail CSRF Security Flaw. 2007. http://ajaxian.com/archives/gmail-csrf-security-flaw.

[7] Fangqi Sun, Liang Xu, and Zhendong Su. 2011. Static detection of access control vulnerabilities in web applications. In USENIX'11: Proceedings of the 20th USENIX Security Symposium.

[8] Prithvi Bisht, A. Prasad Sistla, and V. N. Venkatakrishnan. 2010b. Automatically Preparing Safe SQL Queries. In

FC'10: Proceedings of the 14th International Conference on Financial Cryptography and Data Security.

[9] 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 12th International World Wide Web Conference (WWW'04), pages 40–52, May 2004.

[10] N. Jovanovic, C. Kruegel, and E. Kirda. Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities. In Proceedings of the IEEE Symposium on Security and Privacy, May 2006.

[11] N. Jovanovic, C. Kruegel, and E. Kirda. Precise Alias Analysis for Static Detection of Web Application Vulnerabilities. In Proceedings of the ACM SIGPLAN Workshop on Programming Languages and Analysis for Security (PLAS'06), June 2006.

[12] Y. Xie and A. Aiken. Static Detection of Security Vulnerabilities in Scripting Languages. In Proceedings of the 15th USENIX Security Symposium (USENIX'06), August 2006.

[13] Z. Su and G. Wassermann. The Essence of Command Injection Attacks in Web Applications. In Proceedings of the 33rd Annual Symposium on Principles of Programming Languages (POPL'06), pages 372–382, 2006.

[14] R. Paleari, D. Marrone, D. Bruschi, and M. Monga. On race vulnerabilities in web applications. In Proceedings of the 5th Conference on Detection of Intru-sions and Malware & Vulnerability Assessmen t, DIMVA, Paris, France, Lecture Notes in Computer Science. Springer, July 2008

[15] W. Halfond and A. Orso. AMNESIA: Analysis and Monitoring for NEutraliz-ing SQL-Injection Attacks. In Proceedings of the International Conference on Automated Software Engineering (ASE'05), pages 174–183, November 2005

[16] A. Christensen, A. Møller, and M. Schwartzbach. Precise Analysis of String Ex-pressions. In Proceedings of the 10th International Static Analysis Symposium (SAS'03), pages 1–18, May 2003

[17] C. Gould, Z. Su, and P. Devanbu. Static Checking of Dynamically Generated Queries in Database Applications. In Proceedings of the 26th International Con-ference of Software Engineering (ICSE'04), pages 645–654, September 2004.

[18] R. A. McClure and I. H. Kr¨uger, "Sql dom: compile time checking of dynamic sql statements," in Proceedings of the 27th international conference on Software engineering, ser. ICSE '05, 2005, pp. 88–96.

[19] K. Kemalis and T. Tzouramanis, "Sql-ids: a specification based approach for sql-injection detection," in Proceedings of the 2008 ACM symposium on Applied computing, ser. SAC '08. ACM, 2008, pp. 2153–2158.

[20] P.Grazie, "Phd sqlprevent thesis," Ph.D. dissertation, University of British Columbia(UBC) Vancouver, Canada, 2008.

[21] M. Cova, D. Balzarotti, V. Felmetsger, and G. Vigna, "Swaddler: An approach for the anomaly-based detection of state violations in web applications," 2007.

[22] Weinberger, Joel, Prateek Saxena, Devdatta Akhawe, Matthew Finifter, Richard Shin, and Dawn Song. "A systematic analysis of xss sanitization in web application frameworks." In Computer Security–ESORICS 2011, pp. 150-171. Springer Berlin Heidelberg, 2011

[23] S. W. Boyd and A. D. Keromytis, "Sqlrand: Preventing sql injection attacks," in In Proceedings of the 2nd Applied Cryptography and Network Security (ACNS) Conference, 2004, pp. 292–302.

[24] 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, pp. 106–113.

[25] 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.

[26] 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, pp. 604-611.

[27] Doupé, Adam, Marco Cova, and Giovanni Vigna. "Why Johnny can't pentest: An analysis of black-box web vulnerability scanners." In Detection of Intrusions and Malware, and Vulnerability Assessment, pp. 111-131. Springer Berlin Heidelberg, 2010

[28] Li, Xiaowei, and Yuan Xue. "BLOCK: a black-box approach for detection of state violation attacks towards web applications." In Proceedings of the 27th Annual Computer Security Applications Conference, pp. 247-256. ACM, 2011

[29] Felmetsger, Viktoria, Ludovico Cavedon, Christopher Kruegel, and Giovanni Vigna. "Toward automated detection of logic vulnerabilities in web applications." In USENIX Security Symposium, pp. 143-160. 2010

[30] Guha, Arjun, Shriram Krishnamurthi, and Trevor Jim. "Using static analysis for Ajax intrusion detection." In Proceedings of the 18th international conference on World wide web, pp. 561-570. ACM, 2009.