# Positive Inspection on Web Application Security Attacks and Vulnerability Detection: SQLI

K.Naveen Durai, Assistant Professor
*Dept. of Computer Science and Engineering*
*Sri Eshwar College of Engineering Coimbatore, Tamilnadu, INDIA*

Dr.K.Baskaran, Associate Professor
*Department of Electrical and Electronics Engineering*
*Govt. College of Technology Coimbatore, Tamilnadu, INDIA*

Abstract- **The emergence of the internet in the current decade is enormous and irresistible. The internet users are passionate and swelling their impulsion in making the state of affairs still down reaching. Their bibliography has been totally exaggerated. Every minute the web database is hold no more up with huge amount of information. Thus prolonged habituation sensitized in passwords, credit card numbers, company statistics etc. As a pessimistic and optimistic collaboration, the attackers make reimbursement with naive information. Beyond the shadow of users are threatened with malfunctions. Besides the robust embossed Web security mechanisms, culpability glides that tracker avail oneself of to threaten the information accumulated in the web database. SQL injection (SQLI) and cross site scripting (XSS) are the vital prey feeding expanse for the web security threats. Proposal on recognize and reveal are titanic in submission; recognizing the SQLI and revealing the vulnerabilities in the web application. In this paper, assorted advents**
**be pertinent to SQLI detection and SQLI prevention are scrutinized and compared.**

Keywords – **Input sanitization, SQL injection, Content spoofing, Hot Query Bank, Cross site scripting, Vulnerability.**

## I. Introduction

How does this SQLI code works? This is literally mentioned as the code injection technique and fully fledged in illegitimately ingress into the database. The input used to achieve their objective is as inserting characters or command and that could restrict the query execution. Further, proceeding with the query logic change lets room to the malevolent exploitation in database. The mentioned is a benchmark in a web application where SQL statements are mentioned to entree the database.
What synthesizes the leeway for a web application to be hacked? It is the mainly the novice developers. For exemplification, there is much sophistication in the developing programs (E.g.: java, ASP.NET and PHP) at some point in the construction and execution of SQL statements. These manifest hacking occurs due to improper confirmation and authentication.

Concatenating strings that built the dynamic queries in which most of the web applications depends on. Before the execution of the queries, users have to initially enter in the different input sets through which a query sets will be generated and executed. This facilitates the hackers much and more. All this has taken place in wide range as we are using the dynamic pages. To avoid the web application to be the reason behind this SQLI attacks, the better solution is using the stored procedure instead. It has own shortcoming of inappropriate usage will encourage them to be violated.

## II. WEB APPLICATION ARCHITECTURE

We are all familiar in an aspect that the web application is the one that is running in a web browser as a program. It basically runs under a three-tier construction. As per the request from the browser the presentation tier is shown in Fig., 1 that is sent to a web browser.

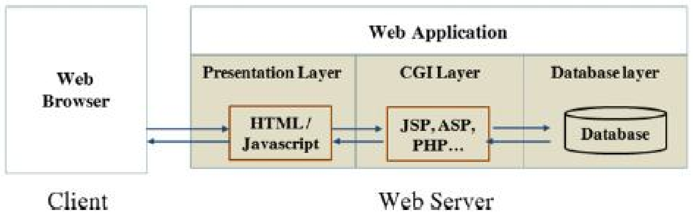

Figure 1.Web Application Architecure

*1) Presentation Tier:*

This is presumed as a Graphical User Interface (GUI). HTML, Flash, Java script, etc are those that directly coordinates and communicates with the user and are the part of this presentation tier. The users input are received and the processed output is displayed to the users. Web browser analyzes this tier.

*(2) CGI Tier:*

The main agenda of this tier is to process the input data and propel the result into the database tier. This is also mentioned as Server Script Process. This is intermediate level tier process as it placed in between the presentation and database tiers. The actual process is, stored data are sent from the database tier to this phase of the tier and again the data are passed on to the presentation tier for users review. This is an all rounder phase where all the data processing of the web applications are taken place. Supported languages are: JSP, PHP, ASP, etc.

*(3) Database Tier:*

The task of this last phase is, storing, managing and retrieving all the data of a web application. Their main drawback relies upon the transformation of data from this phase to CGI tier phase is carried over without any security check. This gives more possibilities in malfunctions as data revealing is trouble-free once CGI Tier tracking is succeeded.

## III. SQLI CAUSES

*A. Absence of checks*

There might be several reasons for the SQLI and one of the most prominent among those is usually the inputs used in the SQL statements that too without checking. Whenever the input is supposed to be checking in regards to its relevancy. Else, there comes an opening for the attackers who could easily harm and hack using the hazardous character inputs that are certainly irrelevant to the input that we are looking for.

```
$query = " SELECT info FROM user WHERE name = ' $_GET[ " name " ] ' AND pwd = ' $_GET[ " pwd " ] ' " ;
```

Here reflect on a case, when the input parameter name value is noted as x’ *OR ‘ 1 ’ = ‘ 1*, when the WHERE clause condition is,

“WHERE name = ‘ x ’ OR ‘ 1 ’ = ‘ 1 ’ AND … ”

This leaves room to the hackers to hack the user information and also without valid account. TRUE will be the evaluation status and the attackers will retain access portability.

*B. Insufficient data type checks.*

Meanwhile, whilst the construction of the SQL statements, it is foremost prominent that a developer must be in utmost concern in checking the data types and the absence of doing this will absolutely end up in a vulnerable web application. Here is a sample elaboration. To overcome all this shortcomings, developers are most probably making use of sum of the sophisticated functions that are used in such every programming languages, such as sanitization functions (sanitizes the input parameters even before used in SQL statements). However, such usages become nullified once the input that are to be accessed are: non-text-based data types or numeric data and there occurs a necessity for is_numeric() functions for checking principles and thus preventing SQLIAs.

*C. Absence of proper delimiters.*

Delimiters are the core to pin point the input's data type, so they are supposed to be used properly while a query is with input strings and are constructed dynamically. Sample shown below is about SQL statement that indicates input string but the case is that the delimiters are not included here in this PHP code

```
$name = mysql_real_escape_string ($_GET [ “ name ” ]);
$query = “ SELECT info FROM user WHERE name = $name ” ;
```

Here for instance, the attacker will be able to easily deceive the input sanitization functions using the alternate coding, when the database server automatically enables the automatic type conversion function. Consider, when the hackers encode into the parameter “name” the HEX string as 0x270x780x270x200x4f0x520x200x310x3d0x31, it will be instantly converted as “varchar” value (result: string ‘ x ’ OR 1=1) in the database parser and this will be never and ever be detected by any of the program's escaping function(s) as they are special characters encoded in the HEX string

*D. Improper parameterized queries or stored procedures.*

We may also discuss about one more criteria that many are not even aware of. That is, SQL injection may be pull off with non-parameterized inputs; when the stored procedures or parameterized query strings acknowledge them.

Shown down is a PHP code:

```
$query = “ SELECT info FROM user WHERE name =
? ” . “ ORDER BY ‘ $_GET [ “ order ” ] ’ ” ;
$stmt = $dbo->prepare ($query);
$stmt->bindParam(1, $_GET[ “ name ” ]);
$stmt->execute ();
```

Thus taking into account, when an attacker could not precede the SQLIA with the parameter “name” he/she may sustain it through parameter “order”, as it is not parameterized. ASC, DROP TABLE user are the query that reveal the hackers to infuse malicious queries to the original ones.

## IV MITIGATING SQL INJECTION ATTACKS

To mitigate the SQL injection attacks, there are 10 ways. They are:

1. Suspect everything: never be culminated. Always verify and sanitize any of the user-submitted data as if they are iniquities.
2. Avoid dynamic SQL wherever possible: Instead make use of stored procedures, prepared statements or parameterized queries wherever applicable.
3. Patch up and update: once we practically apply the update and patching, the hackers will not be easily able to hack the database using their vulnerabilities in the applications. As they are highly abused using SQL injection.
4. Firewall: this is one of the best remedy to filter the calumniated data. The application firewall (WAF) may be appliance or software based. Considerably when they are good, new data could be added wherever likelihood appears and they might always possess a complete set of rules that are parallel default. Even

before the patch available it becomes the apt responsibility of the WAF to inherit security against the vulnerabilities.

5. Attack surface reduction: have some considerations. If you could cork surely in a preference to avoid some of the database functionality from prevention, then go for it. Consider an example, a hacker could be gained indeed of the xp_cmdshell extended stored procedure that are abbreviated in the MS SQL as it generates a Windows command shell and during execution they passes a string either. Whereas, the privileged security is same in both Windows process and SQL Server service account.
6. Proper privilege usage: this is of course a very valuable criterion. Ever and never unnecessarily entail your admin-level privilege until and unless a need arises. As this could obviously limit the enhancement of the attackers themselves.
7. Despite of all; “Secret”: maintain certain scenarios intensely. When you hash the connection strings, other off the record data and passwords always remember to keep a word in yourself that your application is totally unsafe.
8. Be precisely informative: the error messages are the actual prey feeders for the hackers who could effortlessly knot up with database architecture. Minimum information is preferably prescribed. To fool out with the attacker, it is advisable to endorse the usage of the "RemoteOnly" customErrors mode (or equivalent) that will show up the error messages in the local machine.
9. Basics are ought unforgettable: password change is extensively recommended when it is done regularly as this is one of the practically safe and sensitive ways.
10. Better software purchase: before software delivery, it is the utmost conscientiousness of the code writer to check out following two submissions. I) security flaws in the customized application are to be fixed and ii) codes are to be checked perfectly.

## V Methodologies used in detecting SQLI

### *A. Static analysis*

In the Static analysis mentioned here, it is the mere responsibility of the SQL query statements to be analyzed and in that way it could spot and put a stop to SQLI. Its process flows out in such a way that: i) scanning the application, ii) analyzing the information flow and iii) detecting the code segment that might cause harm to the SQLI. If any of the code is found to be vulnerable then that code and the web application are to be rewritten. But the authentic and crucial content of the Static analysis relies upon certifying the user input type and reducing the percentage of SQL injection attacks and not to detect them. Taking into consideration, JDBC-Checker [1] is probably making use of the Java String Analysis (JSA) library. Its functionality is dynamic validation of user input and preventing the
SQL injection hits. This might not be the case when the attacker himself/herself could give accurate input data type or syntax. One of the familiar cores that use static analysis method is Wassermann [2] and it is mingle of automated reasoning. This method is not up to the mark in the SQL injection attack detection and its background is on tautologies.

### *B. Dynamic analysis*

The operation task of Dynamic analysis is; analyzing on every input has been undertaken on the web application and its http response. This analyzing process is termed out as scanning. Scanning is a prolonged process that sends out all the possible input into the target and thus receives the responses. The first and foremost privileged aspect of this dynamic analysis is that, even when none of the changes are accumulated to the web application it will explicitly locate the vulnerable code segments. SQLI attacks are protected using Sania [3] as:

1) Possible vulnerabilities are analyzed using a static approach. Primarily, the normal SQL queries are collected between the client and web applications and web application and database.
2) Vulnerable code segment cab be disclosed and established through the generation of SQLI attack codes.
3) Once being attacked by the SQLI attack codes the generated SQL queries are collected.
4) Comparison is been carried out between the normal and attacked SQL queries.

5) The queries that are been attacked is found to be succeeded or not only after analyzing them through Parse tree.

From the determinations it well implicated that this sounds better than the HTTP response method. Nevertheless, they are all predefined. When new SQLI attack types are exposed they become incapably inefficient. This method becomes advantageous because web application code will not even require any modifications. Thus concluded and convinced that the vulnerabilities are something that is ought to be fixed manually and not through predefined attack codes.

### *C. Hybrid approaches*

There comes another approach where both the static and dynamic approaches are clubbed up altogether and they are termed out as hybrid approach. This pattern is boomed out to become a conclusion for the shortcoming attained in the above two approaches. But it is enduringly and everlastingly considered to be inefficient as of the exorbitant utilization of system resources.

### *D. Hot Query Bank approach*

In the eternity to improvise the ability of the SQLI detectors, [4] a terminology is introduced so called hot query bank. The detected queries that are analyzed and legitimately identified are been recorded here. This results in the trouble-free factor that, once a set of genuine queries are built it then the process of checking the queries becomes nullified thus saving the system resources and improvising the system performance

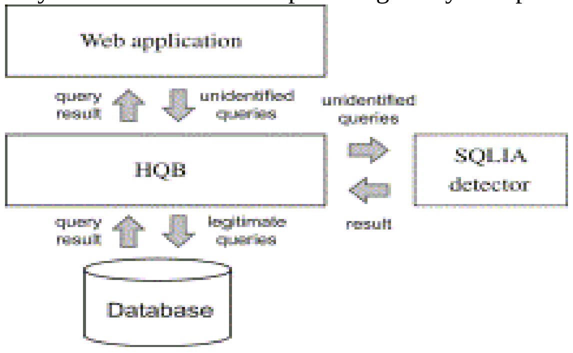

Figure 2. Hot Query;s Sytem Architecture

Above mentioned (Fig.2) is the hot query’s system architecture. It illustrates that constantly the execution query will be sent from the web application to the HQB. Then the previously mentioned checking process (hot query) will be performed. If the query is already proved to be legitimate, they are then executed into the database. If the query is coming up for the first time, then their legitimacy is checked by the SQLI detector and the process will be taken place in the HQB as per their output.

This also poses some of the negative aspects: i) malicious query(s) ratio ii) hot queries updating time iii) sliding window’s size and iv) system’s performance is notably effected through the support parameter. Different detectors have shown the percentage of 45 as the improvised system performance. Bearing in mind the downside, these bank supports only the before now surviving SQLI detectors as they are in necessitate of system performance progression and they do not work on the standalone detectors. In addition, real web application environment has ever and never validated this design and hence it entails a standard interface for collaboration.

### ***E.*** *Removing SQL query attribute value*

Removing the SQL query attribute value for SQLI detection by merging the static and dynamic analysis is a novel method suggested in [5]. Most probably the dynamic SQL queries are generated using the input submitted by the user and they are declared as the attribute value. Function f does the task of removing the SQL query attribute value and it is applicable to both the static (by now examined legitimate queries) and dynamic (formed dynamically through attribute values) queries. Though the dynamic queries are the ones that are to be examined very precisely; function f is applicable to both. When the SQL query attribute values are removed, a logical exclusive operation will be executed and the result will be considered to be relevant for the dynamic query when it is zero. Else, the dynamic query will not be considered as legitimate anymore. This is one the accurate, effective and simple method in detecting SQLI attacks. Besides, cross site scripting (XSS) cannot be perceived through this method.

*F. Mining input sanitization pattern*

The incompetency that is presented in the input sanitization mechanism is compensated through a new method and is suggested in [6]. Input sanitization code patterns are characterized through a set of static code attributes and this is proposed via this method. Reflection of the static code attributes are constructed in the due course of the vulnerability prediction model. Sink i.e., a set of query statement are classified as vulnerable or non-vulnerable using the classification algorithm.

*Prediction model: i) Data preprocessing*

Totally 21 static code attributes (18 are numeric) is used in this particular method. The min-max method that relies in maintaining the attribute range to be within zero to one [0, 1] for normalization is executed as the numeric attributes will always possess different data distribution. Thus, no biasing will be undergone by the classifiers headed for some of the attributes.

*ii) Data reduction*

Sanitization code pattern is something to be distinguished and this can be completed only through Data reduction as they advance the attributes relevancy. Whilst comparing and considering the characterizing the code pattern using the static attribute need not be most effective. Instead, it could be redundant and irrelevant. Now how could one rank the relevancy considering the features? This is done using the chi-square, gain, symmetrical uncertainty and gain ratio. This data reduction enhances the performance of the classifier further.

*iii) Classification*

These classifiers are absolutely sophisticated ones as it undergoes Multi-layer perception and this is done immediately after data preprocessing and data reduction. This is deeply brainy as in the IQ of a human and an outstanding classification done in the sink where the vulnerable or non-vulnerable to the SQLI are classified. These coded are detected in the statement level and not in software level or program level as other detectors. Spaced out from that, the static code attributes are collected easily and it seems to be its tremendous advantage. This could be segmented as an interchange for any of the SQLI detectors or else this input sanitization mechanisms tie up with them. The false alarm rate is tremendously low when a test have been undertaken in eight different web applications Across application is not predicted in this method.

*G. Intelligent detection using multi agent system architecture*

Blocked SQLI are detected through data mining and this multi agent system architecture is suggested in [7]. 'Layers' this is structure of the functionalities here and it is totally a distributed and hierarchical strategy. Speaking about the task; visualization, data gathering and data classification are the functions that are performed through one or many agents in each and every layer. The two available key agents to spot out and slab in the SQLI are: i) a visualize agent and ii) a classifier agent

**i. Classifier Agent**

Case based reasoning mechanism is integrated here. The SQL query statements that are purely in the relation with the SQLI attacks exhibits its history and experience retained. This is termed out as a case. The three different that a case may acquire are: i) initial problem in the web application is described ii) thus obtained problem should possess a solution right? And that is obtained through 'solution' where there are sequential steps as the recover to the problem described and iii) finally step is showing off the state once the problem has been recovered.

Following are the steps in a case based reasoning
1. Retrieve

Retrieving from the memory; Query category attributes identifies the relevancy immediately after the advent of the SQL query. Models that are related to the relevancy are recovered only based on the identified cases.
2. Reuse
i) Retrieved cases and ii) recovered models are the two inputs activation function *f* is used by neural networks where the two inputs are given. Output is normalized and it ranges from [0.2, 0.8].
0.2 - non-attack
0.8 - attack
3. Revise
The suspicious case is evaluated when the output values relies in the range between [0.35, 0.6]. Human expert system reviews all such cases and detection mechanism's precision is improvised.
4. Retain
Bearing in mind about further classification the newly classified cases are updated and reconstructed here.
5. Visualizer agent
The human expert system that is involved in analyzing the apprehensive queries should be more sophisticated and hence the purpose of the visualizer agent. The classification of SQLI attacks is completed here and it is the primary task undertaken into account.
Necessitation for the visualization techniques

- Logical and spontaneous analyzing of data is commenced in Visualization.
- Simplified representation becomes promising even for highly noisy, heterogeneous and complex data through Visualization.
- Visualization affords the fastest detection mechanism.

Formerly classified queries are visualized and tracks out the highly similarity with the suspicious query and this trap is performed by the visualizer agent. This would help further accurate classification. The robustness, flexibility and accuracy of SQLI detection mechanism is improved with means to the combination of the classification algorithm that are used in the visualizer agent and case based reasoning. To err is human. So, wherever suspicious queries are analyzed they are damn enough possibilities of error in the detection process. As the process is not thoroughly automated they are concluded to be unsuitable for dynamic analysis.

## VI. CONTENT SPOOFING

Though the above illustrated methodologies act very well in detecting and preventing the cross site scripting (XSS) and SQLI, they are not designed in a way to work out for content spoofing. Content spoofing, is also phrased as virtual defacement or content injection. This is even well-defined while termed out as attack; as this targets the user and using the injection vulnerability that exploitation is done in the web application. All the mischievous facts are achieved only when an application fails in handling the data supplied by the users. It is then the hackers who enters into the scene and take in hand the worse situation and make use of it by supplying the malicious data to the user with the name of the adorable domain. All this can be abbreviated through passing the parameter value.

## VII. COMPARISON OF VARIOUS SQLI DETECTION METHODOLOGIES

Table -1 Comparison Table

| Parameters/methods | Hybrid approach | Hot Query Bank | SQL query attribute removal | Mining input sanitization pattern | Multi agent system architecture |
|---|---|---|---|---|---|
| Whether Uses machine learning? | X | X | X | ✓ | ✓ |
| Is Modification of code necessary? | X | X | ✓ | X | X |
| Can be used as a standalone | ✓ | X | ✓ | ✓ | ✓ |

| | | | | | |
|---|---|---|---|---|---|
| detector? | | | | | |
| Does Improve performance? | ✓ | ✓ | X | X | ✓ |
| Does Improve accuracy? | X | X | ✓ | ✓ | ✓ |
| If Uses visualization techniques? | X | X | X | X | ✓ |
| Can detect XSS? | X | X | X | ✓ | X |
| Does Consider content proofing attack? | X | X | X | X | X |

## VIII CONCLUSION

This paper, illustrates the survey that brief out the floating methodologies that are used in the SQLI detection. As portrayed, they are detailed in analyzing the merits and demerits. HQB elaborates the definition that it is one outstanding detectors as it possesses increases performance, however, the classifier accuracy is not increased. Though the detectors concentrate on the aspects such as: i) SQL query attribute removal, ii) multi agent system architecture and iii) mining input sanitization pattern to improvise the classifier accuracy, they fall short in detail analysis about the SQL query which leaves room in the deficiency of the detector performance. All these compensations are nullified when the parenthesis are content spoofing. In the upcoming days, we are planning to assimilate both visualizer agent and the HQB and data mining will be used as a method in SQL query classification.

This research succession will deliver us an implementation of outstanding detector in a way or other. This will be to the image for find out watermark image.

## REFERENCE

[1] C. Gould, Z. Su, P. Devanbu, JDBC checker: a static analysis tool for SQL/JDBC applications, in: Proceedings of the 26th International Conference on Software Engineering, ICSE, 2004, pp. 697–698.

[2] G. Wassermann, Z. Su, An analysis framework for security in web applications, in: Proceedings of the FSE Workshop on Specification and Verification of Component-Based Systems, SAVCBS, 2004, pp. 70–78.

[3] Y. Kosuga, K. Kernel, M. Hanaoka, M. Hishiyama, Y. Takahama, Sania: syntactic and semantic analysis for automated testing against SQL injection,in: Proceedings of the Computer Security Applications Conference 2007, 2007, pp. 107–11.

[4] Yu-Chi Chung, Ming-Chuan Wu, Yih-Chang Chen, Wen-Kui Chang, A Hot Query Bank approach to improve detection performance against SQL injection attacks, Elsevier, Computers & Security Volume 31, Issue 2, March 2012, Pages 233–248.

[5] Inyong Lee , Soonki Jeong , Sangsoo Yeo, Jongsub Moon, A novel method for SQL injection attack detection based on removing SQL query attribute values, Elsevier, Mathematical and Computer Modelling, Volume 55, Issues 1–2, January 2012, Pages 58–68.

[6] Lwin Khin Shar , Hee Beng Kuan Tan, Predicting SQL injection and cross site scripting vulnerabilities through mining input sanitization patterns, Elsevier, Information and Software Technology, Volume 55, Issue 10, October 2013, Pages 1767–1780.

[7] Cristian I. Pinzón , Juan F. De Paz , Álvaro Herrero , Emilio Corchado , Javier Bajo , Juan M. Corchado , idMAS-SQL: Intrusion Detection based on MAS to Detect and Block SQL injection through data mining, Elsevier, Information Sciences, Volume 231, 10 May 2013, Pages 15–31.

[8] W. G. Halfond, J. Viegas, and A. Orso, “A Classification of SQLInjection Attacks and Countermeasures,” in Proc. of the International Symposium on Secure Software Engineering, March 2006.

[9] Verizon 2010 Data Breach Investigations Report, “http://www.verizonbusiness.com/resources/reports/rp 2010- databreach- report en xg.pdf.”

[10] Web Application Security Statistics, http://projects.webappsec.org/w/page/13246989/ “Web Application Security Statistics.”

[11] M. Cova, V. Felmetsger, and G. Vigna, “Vulnerability Analysis of Web Applications,” in Testing and Analysis of Web Services, L. Baresi and E. Dinitto, Eds. Springer, 2007.

[12] S. Chong, K. Vikram, and A. C. Myers, “Sif: Enforcing confidentiality and integrity in web applications,” in USENIX’07: Proceedings of the 16th conference on USENIX security symposium, 2007.

[13] Steve Friedl's Unixwiz.net Tech Tips , http://www.unixwiz.net/techtips/sql-injection.html , “SQL Injection Attacks by Example”

[14] WhiteHat Security’s annual study, http://resources.infosecinstitute.com/content-spoofing/, “Content Spoofing“B. Corona, M. Nakano, H. Pérez, “Adaptive Watermarking Algorithm for Binary Image Watermarks”, *Lecture Notes in Computer Science, Springer, pp. 207-215, 2004*.