J Inf Process Syst, Vol.13, No.4, pp.689~702, August 2017
https://doi.org/10.3745/JIPS.03.0024

ISSN 1976-913X (Print)
ISSN 2092-805X (Electronic)

JOURNAL OF INFORMATION PROCESSING SYSTEMS JIPS

A Survey on the Detection of SQL Injection Attacks and Their Countermeasures

Bharti Nagpal*, Naresh Chauhan**, and Nanhay Singh***

Abstract

The Structured Query Language (SQL) Injection continues to be one of greatest security risks in the world according to the Open Web Application Security Project's (OWASP) [1] Top 10 Security vulnerabilities 2013. The ease of exploitability and severe impact puts this attack at the top. As the countermeasures become more sophisticated, SOL Injection Attacks also continue to evolve, thus thwarting the attempt to eliminate this attack completely. The vulnerable data is a source of worry for government and financial institutions. In this paper, a detailed survey of different types of SQL Injection and proposed methods and theories are presented, along with various tools and their efficiency in intercepting and preventing SQL attacks.

Keywords

Dynamic Analysis, Detection, Prevention, SQL Injection Attack, Static Analysis, Vulnerabilities

1. Introduction

In today's age, the ease of accessibility through web applications has completely revolutionized the traditional view of an office or a company. The data is stored in databases, which can be accessed anywhere and anytime through a network. These databases are built on the basis of Codd's principle, which uses SQL (pronounced as 'sequel') to interact with the external environment. The standard format that is followed for all databases has improved consistency, but at the cost of ease of exploitability. The staggering amount of data present on the Internet has led to security threats. Malicious access to this vulnerable data could cause incalculable financial loss along with irreparable damage to one's reputation.

2. Overview of SQL Injection

Structured Query Language (SQL) is a high level language used in database management systems (DBMS). SQL was originally developed in the early 1970's by Edgar F. Codd at IBM. It allows the user to modify, delete, or to just access data. The 'query' is a unit of execution in SQL that returns a set of rows

Manuscript received October 22, 2013; accepted December 04, 2013; onlinefirst April 20, 2015.
Corresponding Author: Bharti Nagpal (bharti_553@yahoo.com)
* Dept. of Computer Engineering, AIACT&R, Delhi, India (bharti_553@yahoo.com)
** Dept. of Computer Engineering, YMCA University of Science & Technology, Faridabad, Haryana, India (nareshchauhan19@yahoo.com)
*** Dept. of Computer Engineering, AIACT&R, Delhi, India (nsingh1973@gmail.com)

and columns depending on the condition specified in the query. In database driven web applications, SQL statements incorporate user-supplied data or text. If an insertion of user-supplied data is done in an unsafe manner, then the web application becomes vulnerable to an SQL Injection Attack.

SQL injection vulnerabilities and attacks occur between the presentation tier and the CGI tier. Most vulnerabilities are accidentally made in the development stage. The data flow of each tier using malicious input data are as shown in Fig. 1. It depicts the user's authentication step. When an authenticated user enters its ID and password, the presentation tier uses the GET and POST method to send the data to the CGI tier. The SQL query within the CGI tier connects to the database and processes the data.

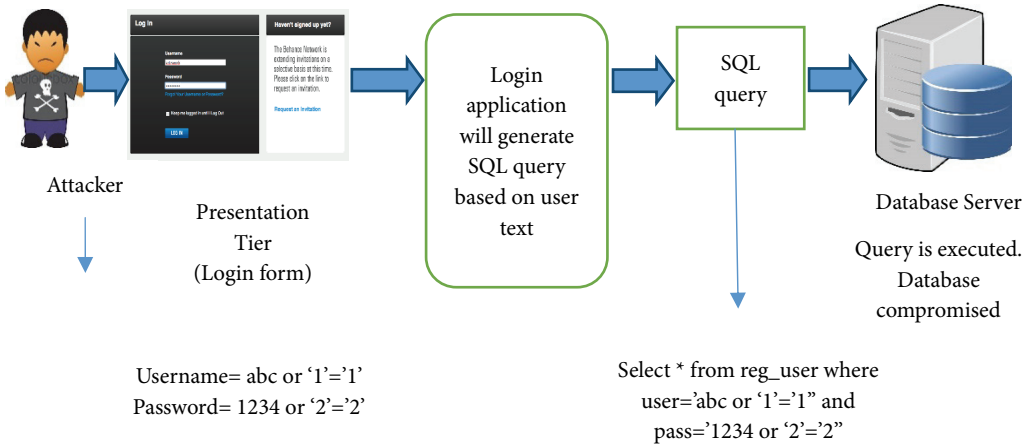

Fig. 1. The data flow of each tier using malicious input data.

3. Types of Vulnerabilities

In this section, the types of vulnerabilities in programming are described in Table 1.

Table 1. Brief description of different types of vulnerabilities

Type of vulnerability	Description
Type 1	Unclear distinction between data types accepted as input
Type 2	Delay of operation analysis till the runtime phase, thus consideration of current variables instead of source code operation
Type 3	Improper type specification while designing
Type 4	Input validation is not well defined and incorrect analysis of sanitized inputs

4. Types of SQL Injection Attacks

In this section, different types of SQL Injection Attacks are described in Table 2.

Table 2. Brief description of different types of SQL Injection Attacks

Type of attack	Attacker's aim	Description	Example
Tautologies	Bypassing authentication and extracting data	Conditional statements are formed in such a way that they are always true.	Select * from emp_info where empid=" or '7=7';
Logically Incorrect queries	To extract information about database and identify injectable patterns	Invalid queries are executed leading to error messages which constitutes information about data type or table name.	Aggregate functions applied on varchar or invalid data types Or using 'having' and 'group by' clauses.
Union Query	Bypassing authentication and extracting data	By using operator 'union', malicious query is joined with safe query.	Select * from user where user='ravi' union select * from admin where id='3142'-- 'pass='2=2';
Stored procedure	Privilege escalation, executing remote commands, DoS	Using built-in procedures, malicious actions are performed.	Commands like DROPTABLE, SHUTDOWN are executed.
Piggy-backed queries	Data extraction and modification, DoS	Malicious query is appended to legitimate query. On execution of first query, second also gets executed.	Select * from user where name= 'ravi' and pass='1234';drop table user;
Alternate Encodings	To evade detection	Some database have filters which detect characters like --, %, etc., as bad character. So to avoid detection, attacker encode the query in ASCII or Unicode.	SELECT salary FROM users WHERE login=" AND pin=0; exec (char (0x73687574646j776e));
Inference	Data extraction, database schema discovery and identification of injectable patterns	Logical conclusions are drawn on basis of true/false questions.	
Blind injection		Database schema is guessed by gathering responses on basis of true/false questions.	Attackers injects query to discover the vulnerabilities like select * from user where id='12' and pass='1=0'; to check if there is input validation or not.
Timing attacks		Information collection is done through observing response time taken in answering questions	Keywords like waitfor are inserted to delay execution if query is true etc.

5. Detecting and Preventing SQL Injection Attacks

In order to prevent SQLIAs many techniques have been proposed. These techniques include defensive coding, encryption and obfuscation, static binding, dynamic binding, etc. The methods and theories proposed vary from introducing middleware to developing completely different detection and prevention algorithms. Concepts from cryptography, machine learning, query translation, etc., have been developed. Listed below are some methods that have gained attention in the past years.

5.1 "Integrated Approach to Prevent SQL Injection Attack and Reflected Cross Site Scripting Attack" [2]

Sharma et al. [2] proposed a query model generator that uses the hybrid approach. In this paper, they proposed a model that is the modification of the existing MHAPSIA model (a model based hybrid approach to prevent SQL injection attack in PHP). This is done by incorporating the logic to prevent SQL injection attack and a proposed algorithm to prevent reflected Cross-site scripting attack.

5.2 "SQLIMW: A New Mechanism against SQL-Injection" [3]

Jiao et al. [3] proposed a new mechanism that defends against SQLIAs by adding a middleware (i.e., SQLIMW) in the system's background. This paper introduced a more efficient and stronger security barrier mechanism in SQLIMW to achieve the safe handling of SQLIMW.

SQLIMW has a double security barrier in similar computation circumstances. The realization strategies are as follows:

- SQLIMW produces a private key, which no one (including system administrators), except for itself, knows.
- In the user information storage system, it stores the user name and password provided. It does so by obtaining the user name, password, and private key for the XOR and perform the hash transformation.
- SQLIMW does the same processing and transformation as in the second step after obtaining the original user data that has been approved by the authentication system. This user data is then submitted to the stored procedure to do the injection detection.

5.3 "Runtime Monitors for Tautology Based SQL Injection Attacks" [4]

Dharam and Shiva [4] proposed a framework to handle tautology based SQLIAs in Java applications by using a post-deployment monitoring technique. This paper extends the framework that was explained in [5]. The basic idea behind their proposed framework is given below.

- Source codes contain certain critical variables that interact with the external world by accepting user inputs, building queries, and processing them by accessing the internal database.
- To monitor the behavior of an application during its execution with respect to an already identified critical variable in order to detect and prevent tautology based SQLIAs.

- When a critical variable violates the checkpoint and in turn follows an invalid path, the runtime monitor immediately detects the abnormal behavior of the application due to the critical variable and notifies the administrator of this.
- This model generates less false positives.

5.4 "An Authentication Scheme for Preventing SQL Injection Attack Using Hybrid Encryption (PSQLIA-HBE)" [6]

Balasundram and Ramaraj [6] proposed an authentication scheme in which they proposed an algorithm that uses both Advance Encryption Standard (AES) and Rivest-Shamir-Adleman (RSA) algorithms to prevent an SQL Injection Attack. In this method, a unique secret key is assigned for every client. On the server side, the server uses a combination of both a private key and public key for RSA encryption. There are three phases, namely, the Registration, Login, and Verification phases. In this method, two levels of encryption are applied to the login query. They are as given below.

- To encrypt the user name and password, symmetric key encryption is used with the help of the user's secret key.
- To encrypt the query, the scheme uses asymmetric key encryption by using the server's public key. The proposed scheme is very efficient as it needs 961.88 ms for encryption or decryption, which is very negligible.

5.5 "TransSQL: A Translation and Validation-Based Solution for SQL-Injection Attacks" [7]

Zhang et al. [7] have suggested a solution in the form of TransSQL to detect malicious SQL queries by distinct environment by using a technique that automatically translates SQL queries into LDAP equivalent queries and validates the results from SQL databases and LDAP databases. TransSQL consists of two phases, which are the preprocessing phase and runtime phase.

- In the preprocessing phase, the information is retrieved from a SQL database to produce corresponding LDAP schema and a LDIF (LDAP Data Interchange Format) file. Then it builds a LDAP equivalent database by importing LDAP schema and LDIF data into the LDAP database.
- In the runtime phase, every SQL request is intercepted between the protected SQL database and a web application and translates it into a LDAP equivalent request.
- SQL request is a SQL injection request if the results returning from both databases have inconsistent responses. After detecting SQL injection request, the result from the SQL database would be replaced with a null result.

5.6 "Effective SQL Injection Attack Reconstruction Using Network Recording" [8]

Pomeroy and Tan [8] have suggested a technique for finding vulnerabilities in a Web application, such as a SQL Injection Attack, by using network recording. In this approach, network forensic techniques and tools are used to analyze the network packets containing the GET and POST requests of a web application. This approach uses a network based Intrusion Detection System (IDS) to trigger the network recording of suspected application attacks.

5.7 "Injection Attack Detection Using the Removal of SQL Query Attribute Values" [9]

Kim [9] presents an effective approach of removing a SQL query passed by the user in SQL query attributes values. This approach uses combined static and dynamic analysis. The proposed method utilizes a function that has the capability to detect the attribute values of a static SQL query in a web application. This function also detects the SQL queries generated at the runtime. This approach profiles the SQL query generated from normal users and compares this with the SQL query that is generated dynamically by the attacker.

5.8 "CANDID: Dynamic Candidate Evaluations for Automatic Prevention of SQL Injection Attacks" [10]

Bisht et al. [10] proposed a tool called Candidate Evaluation for Discovering Intent Dynamically (CANDID), which at each SQL query location dynamically mines programmer intended query structures and detects attacks by comparing intended SQL query against the structure of the actual query issued. Program transformation is used by CANDID to retrofit Web applications written in Java. The proposed algorithm records the programmer-intended SQL query structure on any input (candidate inputs) from the legitimate user and compares this with the query structure generated with the attackers input.

5.9 "Obfuscation-Based Analysis of SQL Injection Attacks" [11]

Halder and Cortesi [11] proposed a method on an obfuscation/deobfuscation-based technique, which is used to detect SQLIAs in a SQL query before sending it to the database. This technique has three phases:

- Static phase: In the static phase, SQL queries in the web application code are replaced by queries in an obfuscated form.
- Dynamic phase: In this phase, user inputs are merged with the obfuscated query during run-time. The dynamic verifier checks the obfuscated query at the atomic formula level to detect a SQL Injection Attack.
- If no SQL injection is found during the verification phase, reconstruction of the original query from the obfuscated query is carried out before submitting to the database.

5.10 "Analysis & Detection of SQL Injection Vulnerabilities via Automatic Test Case Generation of Programs" [12]

Ruse et al. [12] have proposed an approach that uses automatic test case generation to detect SQL injection vulnerabilities. The main idea behind this framework is based on creating a specific model that deals with SQL queries automatically. It also captures the dependencies between various components of the query. In this paper, they used the CREST (Automatic Test Generation Tool for C) test generator and identified the conditions in which the queries are vulnerable.

Based on the experimental results of a few samples, the proposed methodology is able to identify the causal set specifically.

5.11 "Use of Query Tokenization to Detect and Prevent SQL Injection Attacks" [13]

Lambert and Lin [13] have proposed a methodology that consists of tokenizing both the original query and the one with an injection separately. After this is done every token constitutes an index for the array and finally two arrays are formed. Their work consists of implementing a method that detects a single quote, space, or double dashes. All strings before a single quote, before a space, or before double dashes constitute a token. All the tokens grouped together to make an array The tokenization is done for both the original query and the query with an injection. The obtained arrays are then compared and if their lengths differ, an injection is detected, otherwise if their lengths do not differ this means that there is no injection present.

5.12 "Hidden Web Crawling for SQL Injection Detection" [14]

Wang et al. [14] have proposed a hidden Web crawling technology that is based on the access authorization data table (AADT). The Web crawlers can be authenticated and used to gain pages behind login forms by recording authorization information through cookies, sessions, etc. This methodology finds any hidden hyperlinks or forms in respond pages and traverses all of them, thereby detecting any hidden web pages and improving the SQL injection detection. The target of studying the web vulnerability detection mechanisms is to enhance the ability of the web scanner and to raise the Web page coverage of crawler model.

5.13 "A Data-Centric Approach to Insider Attack Detection in Database Systems" [15]

Mathew et al. [15] proposed a methodology to address the problem of threats by an insider by presenting a feature extraction method to model users' access patterns. This paper emphasizes users' access patterns by profiling the *data points* that users' access. This is in contrast to analyzing the *query expressions* that has been used in prior approaches. The data-centric approach is based on the observation that query syntax alone is a poor discriminator of user intent.

5.14 "Combinatorial Approach for Preventing SQL Injection Attacks" [16]

This approach uses both static and dynamic approaches to detect a SQL injection. It is a signature based SQL injection detection technique. In this approach the authors generated hotspots for SQL queries in a Web application code, divided these hotspots into tokens, and then sent these tokens for validation where they used Hirschberg's algorithm, which is a divide and conquer version of the Needleman-Wunsch algorithm, to detect SQL Injection Attacks.

5.15 "An Approach for SQL Injection Vulnerability Detection" [17]

Analysis and Monitoring for Neutralizing SQL Injection Attacks (AMNESIA) is a fully automated technique for detecting and preventing SQLIAs. It works in two phases (i.e., the static and dynamic phases). During the static phase, the model for the different types of queries that an application can legally generate at each point of access to the database is made. During the dynamic phase, queries are intercepted before they are sent to the database and are checked against the statically built models. If the

queries violate the model, then a SQL Injection Attack is detected and further queries are prevented from accessing the database.

5.16 "On Automated Prepared Statement Generation to Remove SQL Injection Vulnerabilities" [18]

Thomas et al. [18] presented an algorithm in which the prepared statements in SQL queries are replaced by secure prepared statements for removing SQL vulnerabilities. These prepared statements have a static structure that prevents SQLIAs from changing the logical structure of a prepared statement. To ensure the efficiency and capability of the proposed algorithm four case studies of open source projects were conducted.

5.17 "Evaluation of Anomaly Based Character Distribution Models in the Detection of SQL Injection Attacks" [19]

Kiani et al. [19] have proposed a same character comparison (SCC) model. In this paper, the authors studied FCD models and tried to overcome their limitation in detecting subtle attacks by proposing a new SCC model. This approach operates by parsing the query section of HTTP requests and creates profiles for each file. The SCC model intercepts HTTP requests and extracts the query section from the request. The training phase is used to determine thresholds, which are applied during the testing phase to identify anomalous requests.

5.18 "Preventing SQL Injection Attacks in Stored Procedures" [20]

Wei et al. [20] have proposed a fully automated technique for detecting and preventing SQLIA incidents in stored procedures. This is done with the combination of static analysis and runtime validation. The basic idea is that the control flow graph of the stored procedures can be represented as an SQL-graph. By using an SQL-graph, they reduced the set of SQL statements. They retrieved a Finite State Automaton (FSA) from the EXEC(@SQL) procedure call and checked SQL statements with the inclusion of user inputs for compliance. They then flagged them as safe or unsafe.

5.19 "A Learning-Based Approach to the Detection of SQL Attacks" [21]

Valeur et al. [21] have proposed an intrusion detection system that uses the machine learning method. The SQL queries generated in a Web application are learned to generate models. The runtime SQL queries are compared with the generated model so as to check for discrepancies.

5.20 "JDBC Checker: A Static Analysis Tool for SQL/JDBC Applications" [22]

Gould et al. [22] have proposed a method in which Instruction-Set Randomization is used. This method inputs random values into the runtime SQL query statement of a web application and checks for volatility, which is used to detect SQLIAs. It places a proxy between the web server and the database server and randomizes SQL queries.

5.21 "Web Application Security Assessment by Fault Injection and Behavior Monitoring" [23]

Huang et al. [23] proposed a Web application security assessment framework called the Web Application Vulnerability Scanner (WAVES). WAVES is a black box testing tool from the research community that can be used to identify Web application vulnerabilities. WAVES uses a Web crawler to find vulnerabilities in Web applications and generates attack codes by utilizing a pattern list and attack techniques. Using the generated attack codes, the SQL Injection Attack vulnerabilities can be found. A brief description of SQL injection approaches proposed by different authors is shown below in Table 3.

Table 3. Description of different approaches given by different authors

No	Author	Description
1	Sharma et al. [2]	A query model generator based on hybrid approach for PHP applications. Provides cent percent detection for known attacks.
2	Jiao et al. [3]	Enforcing authentication by introducing a middleware i.e. SQLIMW in the middle. It implements hash mechanism which is faster than any other encryption.
3	Dharam and Shiva [4]	Injection detection on the basis of identified valid path and critical variables.
4	Balasundram and Ramaraj [6]	RSA and AES encryption used to enforce login query formation.
5	Zhang et al. [7]	Duplicates database and queries into a LDAP database which is platform independent and doesn't propose any change in legacy web applications.
6	Pomeroy and Tan [8]	Network forensics are used to analyse recorded network packets. It uses network based IDS.
7	Kim [9]	Removes attribute values of queries and then dynamic analysis is done on basis of static profile generated.
8	Bisht et al. [10]	Dynamically extracts the query structures from every SQL query location which are intended by the developer.
9	Halder and Cortesi [11]	Queries are obfuscated and then verified, if clean, queries are reconstructed for execution.
10	Ruse et al. [12]	Idea behind this framework is based on creating a specific model that deals with SQL queries automatically.
11	Lambert and Lin [13]	Based on length mismatch of tokenized original and injection queries.
12	Wang et al. [14]	Web crawlers to detect hidden hyperlinks or web pages behind login forms.
13	Mathew et al. [15]	Feature extraction method which models user's access pattern to detect insider's threat to database.
14	Thomas et al. [18]	Proposes secure prepared statement to prevent attempt to change logical structure of queries.
15	Kiani et al. [19]	Same character model which parses query, decides detection on basis of threshold values given to profiles.
16	Gould et al. [22]	Proposes randomization of query by putting random values in runtime query statement to detect injection.
17	Huang et al. [23]	Proposes a web crawler to find vulnerabilities by analyzing the attack and pattern lists through the concept of machine learning.

6. Comparative Analysis

The comparison between different techniques/approaches is shown below in Table 4.

Table 4. Comparison between different approaches/techniques

Technique/approach	Source code adjustment	Attack detection	Attack prevention	Additional infrastructure	Negatives
Hybrid approach	Not needed	Yes	Yes	N/A	Doesn't work for Zero day exploits
SQLIMW	Not needed	Yes	Yes	Middleware	Works for sign-in applications only
Runtime monitors for tautology	Not needed	Yes	Notification generation	Software repository to store source code, critical variables and paths	For Java applications, detects tautology attacks only
Attribute removal (SQL query checker)	Needed	Yes	Yes	Developer learning	Possibility of false positives depending on data in symbol table.
CANDID	Needed	No	Yes	N/A	Performance issues which still need to be resolved.
Obfuscation	Not needed	Yes	Report generation	N/A	Becomes ineffective for SQL queries which are known only at runtime like in Java servlets.
Query tokenization	Not needed	Yes	Blocked query	N/A	Based on length factor only.
Hidden Web crawlers	Updates are required to be done regularly	Yes	Error messages	Additional hardware required to store authorization record	Need an additional mechanism to handle other SQL injection issues.
SQL DOM	Needed	Yes	Yes	Developer Learning	Longer runtime, can't detect Stored procedure type injection
AMNESIA	Not needed	Yes	Yes	N/A	Works for JSP based applications only
Secure prepared statements	Not needed	Yes	Yes	N/A	Becomes vulnerable to iterative statements
SCC model	Not needed	Yes	Alert trigger	Minimal user interaction required	Works effectively for UNION and Obfuscated injections only
WAVES	Not needed	Yes	Report generation	N/A	Cannot detect all vulnerabilities.
Instruction-set Randomization (JDBC checker)	Needed	Yes	Blocked query, further source code adjustment is proposed	Proxy, Developer Learning and Key Management	Becomes ineffective if random value is revealed It doesn't work for illegal or logically incorrect and obfuscated queries.
IDS	Not needed	Yes	Report generation	Training set	Many false positives and negatives in case of poor training set.

6.1 Comparison between Different Approaches and Different Types of SQL Injection Attacks

The comparison between different approaches and different types of SQLIAs is shown below in Table 5 and Fig. 2.

Table 5. Comparison between different types of SQL Injection Attacks versus different approaches

Detection/prevention method	Tautologies	Illegal/incorrect queries	Union queries	Piggy backed queries	Stored procedures	Inference	Alternate encodings
AMNESIA	↑	↓	↑	↑	↓	↓	↓
Tautology checker	↑	↓	↓	↓	↓	↓	↓
Extended MHAPSIA	↑	↑	↑	↑	↑	↑	↑
Attribute removal	↑	↑	↑	↑	↑	↑	↑
CANDID	↑	↓	↓	↓	↓	↓	↓
SQL DOM	↑	↑	↑	↑	↓	↑	↑
Prepared statements	↓	↓	↑	↓	↓	↓	↓
SCC	↓	↓	↑	↓	↓	↓	↑
JDBC checker	N/A	N/A	N/A	N/A	N/A	N/A	N/A
WAVES	⇧	⇧	⇧	⇧	⇧	N/A	⇧
IDS	⇧	⇧	⇧	⇧	⇧	⇧	⇧

↑ Detection/prevention is possible.

↓ Detection/prevention is not possible.

⇧ Partial detection/prevention is possible.

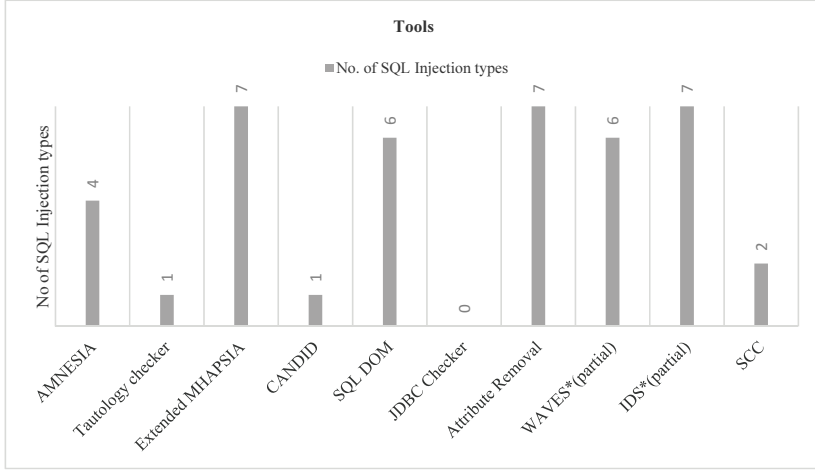

Fig. 2. Number of SQL injection types versus different tools.

7. Conclusion

This paper surveys different SQL injection detection/prevention techniques and tools that have been proposed in the last decade. These techniques vary from enforcing authentication for sign-in web applications to tools that have been developed and proposed to analyze queries for any kind of injection. From the survey of various articles/research papers it has been found that the current techniques are not completely successful. Some have not been implemented yet and some techniques are impractical in reality because they could not address all types of attacks. Keeping in view the emerging web technologies and extensive usage of highly interactive content over the Internet, the developer should follow proper prevention mechanism so as to achieve security against SQL injection.

References

[1] The Open Web Application Security Project, "OWASP Top ten project" https://www.owasp.org/index.php/Category: OWASP_Top_Ten_Project.

[2] P. Sharma, R. Johari, and S. S. Sarma, "Integrated approach to prevent SQL injection attack and reflected cross site scripting attack," *International Journal of System Assurance Engineering and Management*, vol. 3, no. 4, pp. 343-351, 2012.

[3] G. Jiao, C. M. Xu, and J. Maohua, "SQLIMW: a new mechanism against SQL-Injection," in *Proceedings of the International Conference on Computer Science & Service System (CSSS)*, Nanjing, China, 2012, pp. 1178-1180.

[4] R. Dharam and S. G. Shiva, "Runtime monitors for tautology based SQL injection attacks," in *Proceedings of the International Conference on Cyber Security, Cyber Warfare and Digital Forensic (CyberSec)*, Kuala Lumpur, 2012, pp. 253-258.

[5] R. Dharam and S. G. Shiva, "A framework for development of runtime monitors," in *Proceedings of the International Conference on Computer & Information Science (ICCIS)*, Kuala Lumpur, 2012, pp. 953-957.

[6] I. Balasundaram and E. Ramaraj, "An authentication scheme for preventing SQL injection attack using hybrid encryption (PSQLIA-HBE)," *European Journal of Scientific Research*, vol. 53, no. 3, pp. 359-368, 2011.

[7] K. X. Zhang, C. J. Lin, S. J. Chen, Y. Hwang, H. L. Huang, and F. H. Hsu, "TransSQL: a translation and validation-based solution for SQL-injection attacks," in *Proceedings of the 1st International Conference on Robot, Vision and Signal Processing (RVSP)*, Kaohsiung, China, 2011, pp. 248-251.

[8] A. Pomeroy and Q. Tan, "Effective SQL Injection attack reconstruction using network recording," in *Proceedings of the 11th International Conference on Computer and Information Technology (CIT)*, Pafos, 2011, pp. 552-556.

[9] J. G. Kim, "Injection attack detection using the removal of SQL query attribute values," in *Proceedings of the International Conference on Information Science and Applications (ICISA)*, Jeju Island, Korea, 2011, pp. 1-7.

[10] P. Bisht, P. Madhusudan, and V. N. Venkatakrishnan, "CANDID: dynamic candidate evaluations for automatic prevention of SQL injection attacks," *ACM Transactions on Information and System Security (TISSEC)*, vol. 13, no. 2, article no. 14, 2010.

[11] R. Halder and A. Cortesi, "Obfuscation-based analysis of SQL injection attacks," in *Proceedings of IEEE Symposium on Computers and Communications (ISCC)*, Riccione, Italy, 2010, pp. 931-938.

[12] M. Ruse, T. Sarkar, and S. Basu, "Analysis & detection of SQL injection vulnerabilities via automatic test case generation of programs," in *Proceedings of the 10th IEEE/IPSJ International Symposium on Applications and the Internet (SAINT)*, Seoul, Korea, 2010, pp. 31-37.

[13] N. Lambert and K. S. Lin, "Use of query tokenization to detect and prevent SQL injection attacks," in *Proceedings of the 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT)*, Chengdu, China, 2010, pp. 438-440.

[14] X. Wang, L. Wang, G. Wei, D. Zhang, and Y. Yang, "Hidden web crawling for SQL injection detection," in *Proceedings of the 3rd IEEE International Conference on Broadband Network and Multimedia Technology (IC-BNMT)*, Beijing, China, 2010, pp. 14-18.

[15] S. Mathew, M. Petropoulos, H. Q. Ngo, and S. Upadhyaya, "A data-centric approach to insider attack detection in database systems," in *Proceedings of the 13th International Symposium on Recent Advances in Intrusion Detection (RAID)*, Ottawa, Canada, 2010, pp. 382-401.

[16] R. Ezumalai and G. Aghila, "Combinatorial approach for preventing SQL injection attacks," in *Proceedings of IEEE International Advance Computing Conference (IACC)*, Patiala, India, 2009, pp. 1212-1217.

[17] M. Junjin, "An approach for SQL injection vulnerability detection," in *Proceedings of the 6th International Conference on Information Technology: New Generations (ITNG'09)*, Las Vegas, NV, 2009, pp. 1411-1414.

[18] S. Thomas, L. Williams, and T. Xie, "On automated prepared statement generation to remove SQL injection vulnerabilities," *Information and Software Technology*, vol. 51, no. 3, pp. 589-598, 2009.

[19] M. Kiani, A. Clark, and G. Mohay, "Evaluation of anomaly based character distribution models in the detection of SQL injection attacks," in *Proceedings of the 3rd International Conference on Availability, Reliability and Security (ARES'08)*, Barcelona, 2008, pp. 47-55.

[20] K. Wei, M. Muthuprasanna, and S. Kothari, "Preventing SQL injection attacks in stored procedures," in *Proceedings of the Australian Software Engineering Conference (ASWEC)*, Sydney, 2006.

[21] F. Valeur, D. Mutz, and G. Vigna, "A learning-based approach to the detection of SQL attacks," in *Proceedings of the 2nd International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, Vienna, Austria, 2005, pp. 123-140.

[22] C. Gould, Z. Su, and P. Devanbu, "JDBC checker: a static analysis tool for SQL/JDBC applications," in *Proceedings of the 26th International Conference on Software Engineering (ICSE)*, Edinburgh, Scotland, 2004, pp. 697-698.

[23] Y. W. Huang, S. K. Huang, T. P. Lin, and C. H. Tsai, "Web application security assessment by fault injection and behavior monitoring," in *Proceedings of the 12th International Conference on World Wide Web*, Budapest, Hungary, 2003, pp. 148-159.

Bharti Nagpal

She received her M.Tech. (Information Systems) from Netaji Subash Institute Of Technology (N.S.I.T), Delhi in 2010 and B.Tech. (Computer Eng.) from NIT Kurukshetra in the year 1999. She has about 12 years of teaching experience. Presently, she is working as Assistant Professor in Dept. of Computer Eng. at Ambedkar Institute Of Advanced Communication Technology & Research(AIACT&R), Delhi (India). Her research interest includes web technologies, information security, web mining, data mining and data warehousing.

Naresh Chauhan

He received his Ph.D. (Computer Eng.) from MD University, Rohtak (Haryana) in 2008, M.Tech. (Information Technology) from GGS IndraPrastha University, Delhi in 2004 and B.Tech. (Computer Eng.) from NIT Kurukshetra, in the year 1992. He has about 21 years of experience in teaching and Industries. He served Bharat Electronics Ltd. and Motorola India Ltd. Presently, he is working as Professor & Chairman in Dept. of Computer Eng. at YMCA University of Science & Technology, Faridabad (India). His research interest includes Internet technologies, Software Engineering, Software Testing and Real time systems. He has published one book on Software Testing published from Oxford University Press, India (2010).

Nanhay Singh

He received his Ph.D. (Computer Eng.) from Kurukshetra University, Kurukshetra (Haryana) in 2011, M.Tech. (Computer Eng.) from Kurukshetra University, Kurukshetra in 1998. He has about 15 years of teaching experience. He served as Assistant Professor in various prestigious institutes like HBTI, Kanpur(Uttar Pradesh) etc. Presently, he is working as Associate Professor in Dept. of Computer Eng. at Ambedkar Institute Of Advanced Communication Technology & Research (AIACT&R), Delhi (India). His research interest includes web mining, web security, web applications and data mining. He has more than 25 research publications in various national/international journals/conferences.