Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X
IMPACT FACTOR: 6.017

IJCSMC, Vol. 6, Issue. 8, August 2017, pg.5 – 17

Detection and Prevention of SQL Injection Attack: A Survey

Zainab S. Alwan[1], Manal F. Younis[2]

[1]Computer Engineering Baghdad University, Iraq

[2]Computer Engineering Baghdad University, Iraq

[1] zainab_aldahlaky@yahoo.com, [2] manalyounis3@yahoo.com

Abstract— SQL (structure query language) injection is one of threats to the applications, which are Web-based application, Mobile application and even desktop application, which are connected to the database. By implementing SQL injection, attacker can gain full access to the application or database so that it can remove or change significant data irresponsibly. Applications that do not properly validate the user's input make them vulnerable against SQL injection. SQL Injection Attacks (SQLIA) occurs when an attacker is able to insert a series of malicious SQL statements into a "query" through manipulating user input data for execution by the back-end database. Using this type of threats, applications could be hacked easily and steal the confidential data by the attacker. In this paper we present classical and modern types of SQLIA and display different existing technique and tools which are used to detect or prevent these attacks.

Keywords— SQL injection, Database security, PDO, Web application, SQLite.

I. INTRODUCTION

Nowadays, the Internet is becoming a wide-spread information infrastructure. The Utilization and fast progress of the Internet infrastructure has motivated the increasing amount of data stored in a database lately. Increasing the number of users and the heavy dependence on digital information is led to the importance of securing data or information spatially if this information about commercial, corporate businesses, institutions, organizations, numerous financial transactions, health information, personal information and other internet based services. Via Internet all web applications can be accessed using any web browsers that run on any operating systems. Web applications have become the interface that is widely used to retrieve or insert these data or information [1]. The web application with a database that stores important information is one of the targets of the SQLIA, since the databases are absolutely accessible by attacker by injecting SQL queries that are retrieved by web application. As user information is frequently kept in these databases, important information is lost and the security violate.

Furthermore, foundation of smartphones gave rise to a lot of creative technology and gave a new aspect to the use of mobile phones for the users. There are two types of smartphone operating systems commonly seen in the market Android and IOS [2]. Most Android applications need to save data like user settings or large amounts of information about application in files or built-in database, which is called SQLite databases [3]. As SQLite is one type of database, so it is also affected by SQLIA. Thus, it is important to realize that all applications, whether they are web, mobile and desktop applications, connect to the database are targets of SQLIA.

II. DEFINITION OF SQLIA

SQL Injection is one of the most common threats to a database system in which the attacker adds SQL statement to an application form input box, to gain access the resources or make changes to data stored into database. Lack of input validation in applications causes attacker to be successful. In an **SQL Injection attack**, the attacker injects a string input through the application, which changes or manipulates the SQL statement to the attacker's advantage. An SQL Injection attack can harm the database in various ways, such as unauthorized manipulation of the database, or retrieval of sensitive data. It can also be used to execute system level commands that may cause the system to deny service to the application. This issue is very risky because it can cause data loss or misuse of data by parties who are not authorized and as result the functionality and confidentiality are destroyed [4]. Basically, the process of SQLIA is illustrated in Fig.1:

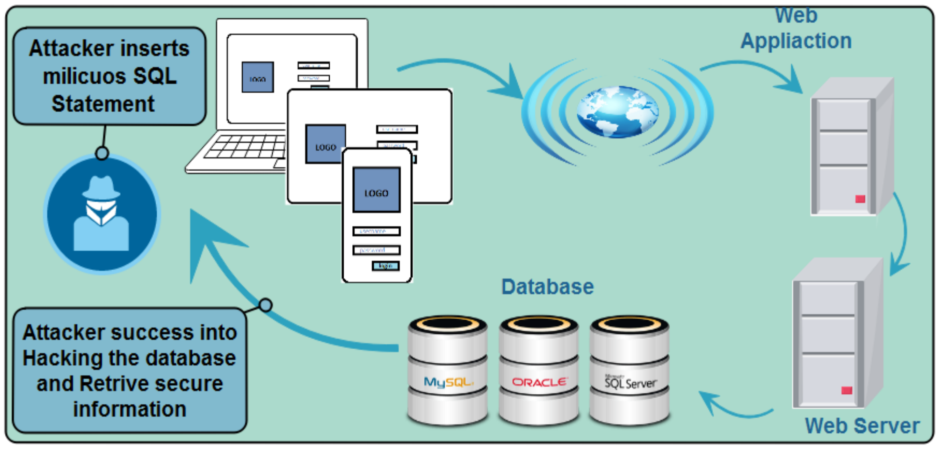

Attacker inserts milicuos SQL Statement
Web Appliaction
Database
Attacker success into Hacking the database and Retrive secure information
MySQL
ORACLE
SQL Server
Web Server

Fig. 1: SQLIA procedure.

III. RISKS ASSOCIATED WITH SQL INJECTION ATTACK

SQL injection is harmful and the risks associated with it provide motivation for attackers to attack the database. The main consequences of these vulnerabilities are attacks on the following characteristics [5]:

a) **Authorization***:* Critical data that are stored in a vulnerable SQL database may be altered by a successful **SQLIA**.

b) **Authentication*:*** If there is no any proper control on input fields inside the authentication page, it may be possible to login into a system as a normal user without knowing the authenticated user.

c) **Confidentially*:*** Usually databases are consisting of sensitive data such as personal information, credit card numbers and/ or social numbers. Therefore, loss of confidentially is a terrible problem with **SQL** Injection vulnerability.

d) **Integrity*:*** By a successful **SQLIA** not only an attacker reads sensitive information, but also, it is possible to change or delete this private information.

e) **Database Fingerprinting**: The attacker can determine the type of database being used in backend so that he can use database-specific attacks that correspond to weakness in a particular database management system [4].

IV. CLASSIFICATION OF SQLIA

There are numerous research papers that present a classical SQL Injection attacks, and Detection and Prevention techniques, while the modern SQL Injection attacks, which are the most dangers, are not presented. Therefore, in the following subsections we will discuss and analysis the modern types of **SQLIA** besides present the classical types.

A. Classical Types of SQLIA

As presented in many papers, the classical types of SQLIA can be listed as follows:

1. **Tautologies:** Tautology-based attacks work through injecting code by one or more conditional SQL statement queries in order to make the SQL command evaluate as a true condition such as **(1=1)** or **(- -).** The most common use of this technique is to bypass authentication on web pages resulting in access to the database. The SQL query shown in Fig. 2 illustrates the tautology **SQLIA** [6].

```
Select * from employees where
employee_ID='1'or '1=1'--'AND
employee_password='1234';
```

Fig. 2: Tautology attack

.

2. **Piggy-backed Query**: Piggy-backed queries is a type of attack that compromises a database using a query delimiter, such as "**;**", to inject additional query statements to the original query. In this method the first query is original whereas the subsequent queries are injected. This attack is very dangerous; attacker can use it to inject virtually any type of SQL command. The SQL query shown in the following figure illustrates the Piggy-backed query attack [6].

```
SELECT pass FROM userTable WHERE
user_Id='user1' AND Password = 0; drop
userTable
```

Fig.3: Piggy-backed attack.

3. **Logically Incorrect**: Logically Incorrect attack takes advantage of the error messages that are returned by the database for an incorrect query. These database error messages often contain useful information that allow attacker to find out the vulnerable parameter in an application and the database schema. The SQL query mention in the following figure describes Logically Incorrect attack [5]**.**

```
SELECT * FROM userTable WHERE
user_Id='1111' AND password='1234'
AND CONVERT (char, no)
```

Fig.4: Logically Incorrect attack.

4. **Union query**: Union query injection is called as statement injection attack. In this attack, attacker insert additional statement into the original SQL statement. This attack can be done by inserting either a **UNION** query or a statement of the form **";< SQL statement >"** into vulnerable parameter, as shown in Fig 5. The result of this attack is that the database returns a dataset that is the union of the results of the original query with the results of the injected query [5].

```
SELECT * FROM userTable WHERE
user_Id='1111' UNION SELECT * FROM
memberTable WHERE member_Id='admin' --'
AND password='1234';
```

Fig. 5: Union query injection attack.

5. **Stored Procedure**: In this technique, attacker focuses on the stored procedures which are present in the database system. Stored procedures run directly by the database engine. it is a piece of code which is exploitable. Stored procedure gives true or false values for the authorized or unauthorized clients. For **SQLIA**, attacker will write **"; SHUTDOWN; --"** with login or secret key. The SQL query, mention in the following figure, will be produced the stored procedure attack [7].

```
SELECT Username FROM UserTable
WHERE user_name= 'user1' AND pass=' ';
SHUTDOWN;
```

Fig. 7: Stored Procedure attack.

6. Inference: Using Inference attack enable the attacker changing the behavior of a database or application. This type of attack can be classified into two well-known techniques, which are: **Blind injection** and **timing attack**.

a) **Blind Injection**: This type of SQLIA occurs when programmers forget to hide an error message which causes the database application insecure, this error message help SQLIA to compromise the database through asking a series of logical questions through SQL statements. The following figure illustrates the blind injection attack [6].

```
SELECT pass FROM userTable WHERE username=
'user' and 1 =0 -- AND pass = AND pin= 0
SELECT info FROM userTable WHERE username=
'user' and = 1 -- AND pass = AND pass= 0
```

Fig. 8: Blind Attack.

b) **Timing Attacks**: This type of attack permits an attacker collects information from a database by observing timing delays in the database's responses. This kind of attack used ***if*** condition statement to achieve a time delay purpose. ***WAITFOR*** is a keyword along the branches, which causes the database to delay its response by a specified time. The following figure illustrates timing attack query [6].

```
declare @ varchar (8000) select @s =
db_name () if (ascii (substring (@s, 1, 1)) &
(power (2, 0))) > 0 waitfor delay '0:0:5'
```

Fig. 9: Timing Attack.

7. Alternate Encodings: This type of attack occurs when attacker modify the injection query via using alternate encoding, such as hexadecimal, ASCII, and Unicode. By means of this way, the attacker can escape from developer’s filter, which scan input queries for special known "bad character". When this type of attack combines with other attack techniques it could be strong, because it can target different layers in the application so developers need to be familiar to all of them to provide an effective defensive coding to prevent the alternate encoding attacks. The SQL query, which is shown in Fig 10, describes the alternate encoding attack [5].

```
SELECT accounts FROM userTable WHERE
login='' AND pin=0; exec
(char(0x73687574646f776e))
```

Fig. 10: Alternate encoding attack.

B. Modern types of SQLIA

In this subsection is presented an overview for modern types of **SQLIA** and mention example for each type, as follows [8]:

1. Fast Flux SQL Injection Attack:

By using this type of SQLIA, attacker intents to extract data from the database and phishing. The phishing is a social engineering attack in which an attacker fraudulently acquires sensitive information from the user by incorporating as a third party. Traditional phishing host can be detected very easily just by tracking down the public Domain Name Server (DNS)or the IP address. This trace back technique could lead to the shutdown of the hosting websites. The attacker realized that conducting like that attack could have significant effect on load balancing of a server; therefore, to protect its criminal assets, the operator of phishing websites started using Fast Flux technique. Fast Flux is a Domain Name Server technique which is used to hide phishing and malware distribution sites behind an ever changing network of compromised host. Fig 11 illustrates Fast Flux SQL attack technique [8].

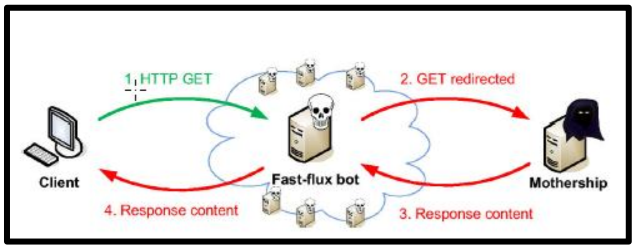


Fig.11: Fast flux attack [8].

2. Compounded SQL Injection Attack:

This type of SQLIA is a combination derived from a conjunction of SQLIA and other Web Application Attacks. The attacker intents to attack the database and cause more serious effect then other classical types of SQLIA (which are previously discussed). The rapid development of prevention and detection techniques against various SQLA, enforced the malicious attackers to develop a technique called compounded SQL Injection. Compounded SQLIA can be detailed as follows [8]:

- **SQL Injection + DDos (Distributed Denial of Service) Attacks:** this attack is used to hang a server, exhaust the resources so that the user cannot able to access it. The different SQL commands which can be used in SQL Injection in order to keep track with DDOS Attack is to encode, compress, join etc. The following figure shows the sample code that is used to perform SQL DDOS attack [8].

```
select tab1 from (select
decode(encode(convert(compress(post)
using latin1),concat(post,post,post,post)),
sha1(concat(post,post,post,post)))
as tab1 from table_1);
```

Fig. 12: SQL Injection + DDos attack.

- **SQL Injection + DNS (Domain Name Service) Hijacking:** By using this type of attack, the attacker intent to embed the SQL query in DNS request and to capture it and makes its way onto the internet. Fig. 13 demonstrates how this attack is performed [8].

```
do_dns_lookup( (select top 1 password
from userTable) + '.inse6140.net' );
```

Fig. 13: SOL Injection + DNS attack.

- **SQL Injection + XSS (Cross Site Scripting):** XSS is the client side code injection attack where an attacker can inject malicious code into the input fields of an application. After inserted the XSS script, it will execute and try to connect with the database of an application. The extraction code for data from the database can be implemented using the iframe command [8] as shown in the following figure.

```
print "<html>"
print "<h1>Most recent comment</h1>"
<iframe src=http://evil.com/xss.html>
print "</html>"
```

Fig. 14: SQL Injection + XSS (Cross Site Scripting) attack.

- **SQLIA using Cross Domain Policies of Rich Internet application (RIA**): Cross Domain policies is an XML file which gives permission to web client application, such as Java, Adobe Flash, Adobe Reader, etc., to access data in multiple domains. Cross-domain policies define the list of RIP hosting domains that are allowed to retrieve content from the content provider's domain [8]. Fig. 14 illustrates Cross domain policy.

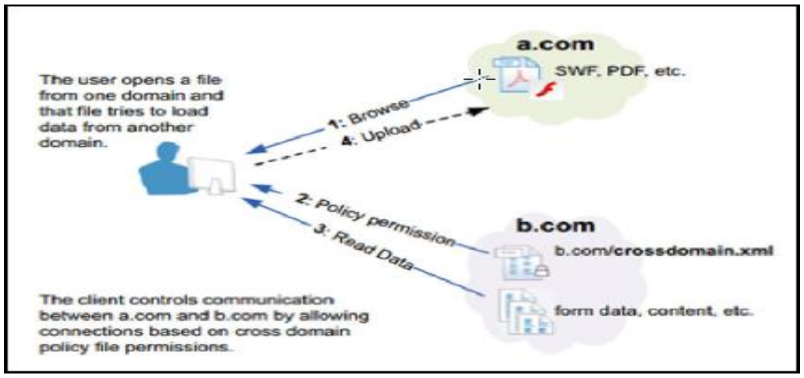

a.com
SWF, PDF, etc.
The user opens a file from one domain and that file tries to load data from another domain.
1: Browse
4: Upload
2: Policy permission
3: Read Data
b.com
b.com/crossdomain.xml
form data, content, etc.
The client controls communication between a.com and b.com by allowing connections based on cross domain policy file permissions.

Fig. 15: Cross domain policy [8].

Fig.16 demonstrates the weak code that makes web application is vulnerable to the SQLIA as well as it is vulnerable to other types of attack.

```
<allow-access-from domain
="*.sub1.domainA.com"/>
<allow-access-from
domain="*.domainC.com"/>
<allow-access-from domain="*"/>
```

Fig. 16: A Week Code for the Cross Domain Policy

- **SQL Injection + Insufficient Authentication**: this type of attack occurs when the user or an administrator is a novice. The security parameters have not been initialized and the attacker can access to the sensitive content without verifying the identity of the user. thereby the attacker exploits this vulnerability to inject SQL injection code. Hence, this type of attack is trouble-free as compared with the other types of attack. The first step is to find whether the application has insufficient authentication and if it has then the SQL Injection attack can take place [8].

V. SQLIA DETECTION AND PREVENTION TECHNIQUES

In the following subsections we will briefly discuss and analysis detection and prevention techniques against classical types as well as modern types of SQLIA as explained below.

A. Detection and Prevention Techniques against Classical Types of SQLIA

In this section we present the most popular techniques, which are used to either detect or prevent classical types of SQLIA while they don't overcome the modern types of SQLIA [8], as follows:

❖ In [9] the author developed a technique, which is called **AMNESIA**, that stands for Analysis and Monitoring for Neutralizing SQLIA. **AMNESIA** combines dynamic and static analysis for detecting and preventing web application vulnerabilities at the runtime. **AMNESIA** uses static analysis to generate different type of query statements. In the dynamic phase, **AMNESIA** interprets all queries before they are sent to the database and validates each query against the statically built models.

❖ In [6], the authors present a new model, which is shown in Fig. 17, to detect and prevent SQLIA at the runtime and it was developed based on SQL syntax-aware at the web application layer, and negative taint at the database layer. Applying negative taint in database layer helps the authors to identify untrusted data at the database layer, while performing syntax-aware evaluation in web application server of query strings, before executing the query in the database gives that model several significant advantages over techniques based on other mechanisms. The advantages of that model apart from efficiency are listed as follows:

- This technique has been successful against all classical types of SQLIAs because the dynamic SQL statement is monitored through the database layer.
- Also it does not change the web architecture.

while the drawback of this model it imposes very low overhead on the system, that it would be determined by the network speed and database server access [6].

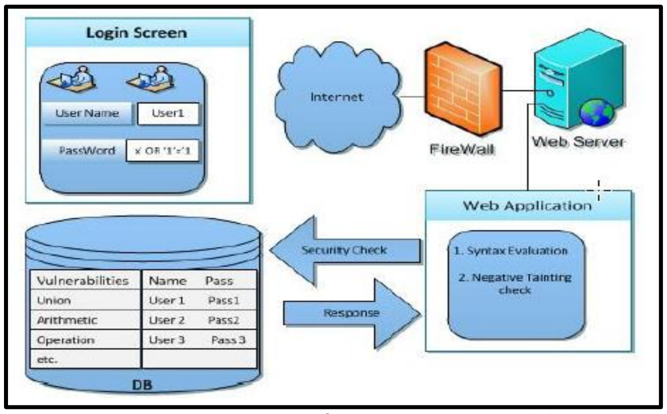


Fig.17: the architecture of the proposed model in [6].

❖ In [10], the authors presented a tool, which is called **SAFELI**, which capable of discovering very delicate vulnerabilities by taking advantage of source code information. SAFELI framework aims at identifying the SQL Injection attacks during the compile-time. The drawback of this tool that the implementation does not complete, and dose not overcome the tautologies attack.

❖ In [11] the authors developed a **WASP** (Web Application SQL Injection Protector) tool which is efficient and effective in stopping more than 12,000 attacks without generating any false positives in non-real-time environment. The limitation of this tool can be founded by implemented the approach for the binary applications and deployed web applications.

❖ In [12], the author developed a **R-WASP** (Real Time-Web Application SQL Injection Detector and Preventer) tool, which capable to stop all attacks effectively and detects SQLIAs in real-time environment. The limitation of this tool is required more practices in order to mitigate stored procedures attacks efficiently.

❖ In [1], the authors developed a suitable Real Time Web Application SQL Injection Protector (**RT-WASP**) tool to detect SQL injection attacks in stored procedures, besides detecting all classical types of SQLIA. The drawbacks of **RT-WASP** tool that it does not detect the SQL Injection and XSS attack, thus, the authors plan to extend **RT-WASP** technique to encompass SQLI and XSS attack in the web applications [1].

❖ In [13], the authors proposed a technique that is based on the principle of dynamic query structure validation which is done through checking query's semantics. This major purpose of this technique focuses on this particular kind of attack, stored procedure attack, along with general prevention.

❖ In [14], the authors implemented a prevention tool for java which is called **SecuriFly**. This tool is used to chase string instead of character for taint information and try to sanitize query strings that have been generated using tainted input. The limitation of this tool are listed as follows [14]:

- This approach cannot use to stop an injection result from inserting SQL command in numeric fields.
- the main limitation of this approach is the difficulty of identifying all the sources of user input.
- Besides, this technique stops all classical types of SQLIA partially because of limitations of the underlying approach [5].

❖ In [15], the authors using a **JDBC-Checker**, which is technique used for statically checking the correctness type of dynamically generated SQL queries. JDBC-Checker is able to detect major causes of SQLIA vulnerabilities in code, which is checking on the improper typing of the query. The drawbacks of this technique are listed as follows:

- this technique could not catch more general types of SQLIA because in the most of these attacks, the attacker writes queries correctly.
- As mention in [5] the **JDBC-Checker** stop all classical types of SQLIA partially.

❖ In [16], the author presents a Dynamic Candidate Evaluations method for automatic prevention of SQLIA, which is called **CANDID**. This technique dynamically extracts the query structures from each SQL query location which are intended by the developer. Hence, **CANDID** solves the issue of manually modifying the application to create the prepared statements. The drawback of this technique is partially stop SQLIAs because of the constraints on the basic approach.

❖ In [17], the author proposed an approach, which is called **Swaddler**, to analyze the internal state of a web application. It works based on both single and multiple variables and shows an impressive way against complex attacks to web applications. Initially, the approach describes the normal values for the application's state variables in critical points of the application's components. later, during the detection phase, the approach monitors the application execution to identify abnormal states. The disadvantage of this approach that it partially detects SQLIAs because of some limitation of underlying approach.

❖ In [18], the authors suggest using a **DIWeDa**, which is a prototype that acts at the session level rather than the SQL statement or transaction stage, to detect the intrusions in Web applications. The proposed framework is efficient and could identify SQL injections and business logic violations too. the drawback of this technique that it does not have the ability to detect all (classical and Modern) types of SQLIA except its ability to detect Inference attack.

❖ In [19], the authors used a **Positive Tainting and Syntax Aware Evaluation** technique that is based on positive tainting and syntax aware evaluation. Where this technique categorizes input strings as hard coded strings, strings implicitly created by Java and strings originated from external sources. Besides that, it propagates the trusted or untrusted markings based on the initialization. Subsequently, a 'syntax aware evaluation' is performed to evaluate the propagated strings. Thereby, based on the evaluation result, if untrusted strings are found, such queries are restricted from passing into the database server for processing, otherwise these queries are passed. According to [19], the major advantage of this technique is prevent all classical types of SQLIAs, while the drawbacks of this technique are listed as follows [19]:

- Initialization of trusted strings are determined by developers.
- Persistent storage of trusted strings may cause modern type of attack like (SQL Injection and XSS attack).

❖ In [20], the authors proposed a technique, which is called **SQL Prevent,** that is consists of an HTTP request interceptor. The HTTP requests are saved into the current thread-local storage. Then, SQL interceptor intercepts the SQL statements that are made by web application and pass them to the SQLIA detector module.

Consequently, HTTP request is fetched from thread-local storage and examined to determine whether it contains an SQLIA. The malicious SQL statement would be prevented to be sent to database, if it is suspicious to SQLIA. **SQL Prevent** detects all type of SQLIA but as mention in [8] it cannot overcome the modern types of SQLIA.

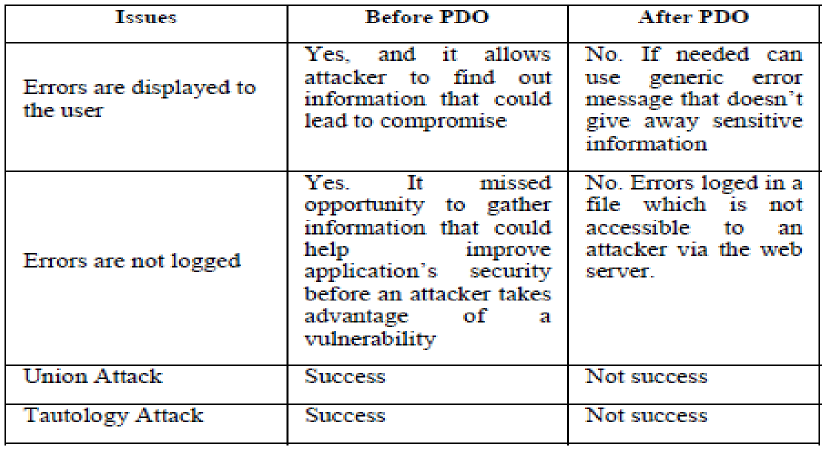
❖ In [21], the author highlights the use of **Automated approaches** in order to prevent SQLI input manipulation flaws. This approach is based on defensive programming and code review, where in defensive programming an input filter is implemented to prevent user from entering malicious keywords or characters, this is achieved by using white lists or black lists. The major advantage of this technique is a low cost mechanism in detecting bugs, while the major drawbacks of this technique are listed as follows [21]:

- It is required deep knowledge on SQLIAs.
- It is unable to detect the stored procedure attack, Alternate Encodings and (SQL Injection and XSS) attack.

❖ In [22], the author proposed a technique, which is called **SQLIPA** (SQL Injection Protector for Authentication), that adopts hash value approach to further improve the user authentication mechanism. There is a hash value for username and other for password. Whenever user account is created, the hash values, that represent username and password, are created and calculated at runtime. The drawback of this technique is the **SQLIPA** has the ability to detect the only Tautologies attack and cannot detect other classical and modern types of SQLIA [22] [8].

❖ In [23] the authors highlight the usage of **prepared Statements**. In case of using the JDBC in database connectivity, the Prepared Statement interface automatically escapes the special characters before executing the query. Therefore, the Prepared Statement helps the developer to prevent SQLIA since it separates the values in a query from the structure of SQL.

❖ In [24], the authors proposed a technique which is implemented **PDO** (PHP Data Object). **PDO** extension defines a lightweight, consistent interface for accessing databases in PHP, and has become one of the trends in developing dynamic web applications that connect to the database. The purpose behind using **PDO** in [24] is security, easy installation, faster execution, and flexibility when connected to the system database. **PDO** Parameterized Queries can prevent SQL Injection attack which is result from tautologies attack and union attack. The evaluation result before and after implementation **PDO** Parameterized query is demonstrated as shown in the following figure:



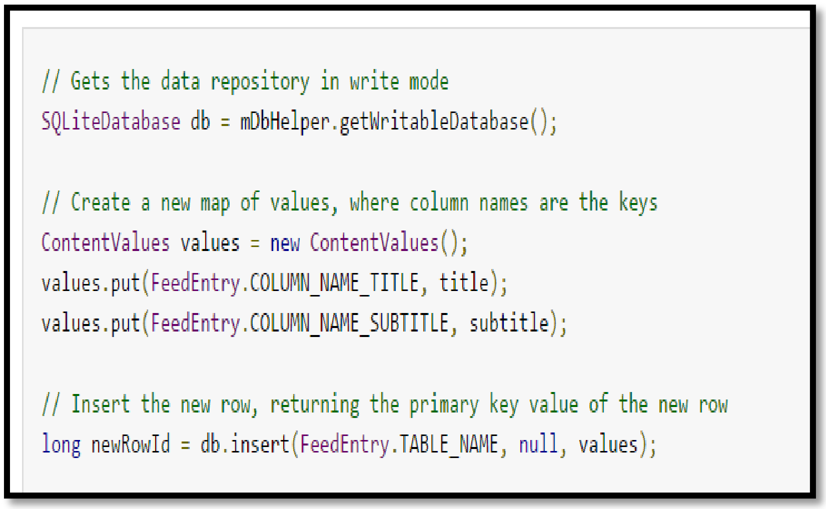
Issues	Before PDO	After PDO
Errors are displayed to the user	Yes, and it allows attacker to find out information that could lead to compromise	No. If needed can use generic error message that doesn't give away sensitive information
Errors are not logged	Yes. It missed opportunity to gather information that could help improve application's security before an attacker takes advantage of a vulnerability	No. Errors loged in a file which is not accessible to an attacker via the web server.
Union Attack	Success	Not success
Tautology Attack	Success	Not success

Fig.18: The results of PDO test [24].

As mention in [24], the main drawback of this technique that it can prevent tautologies attack and union attack only

❖ As mention previously, the Smartphone has built-in database, which is called SQLite, thus smartphone is vulnerable to SQLIA also. Android platform provides full support for SQLite database and the android developers develop more precious database API to provide a mechanism for creating selection criteria that protects against SQL injection. The mechanism divides the selection specification into a selection clause and selection arguments.

The clause defines the columns while the arguments are values to test against that are bound into the clause. Because the result isn't handled the same as a regular SQL statement, it is invulnerable to SQL injection [25]. the following figure shows the snippet that illustrates how to use ***insert*** () function and its argument.



```
// Gets the data repository in write mode
SQLiteDatabase db = mDbHelper.getWritableDatabase();

// Create a new map of values, where column names are the keys
ContentValues values = new ContentValues();
values.put(FeedEntry.COLUMN_NAME_TITLE, title);
values.put(FeedEntry.COLUMN_NAME_SUBTITLE, subtitle);

// Insert the new row, returning the primary key value of the new row
long newRowId = db.insert(FeedEntry.TABLE_NAME, null, values);
```

Fig. 19: snippet for *insert ()* function in Android platform [25].

B. Detection and Prevention Techniques against Modern Types of SQLIA

In this section we will discuss various techniques for the detection and prevention of modern SQLIA as follows:

❖ In [26], the authors presented a Fast Flux Monitor (**FFM**) for real time detection of fast flux service networks based on using both active and passive DNS monitoring. The proposed approach uses active and passive sensors derived from DNS monitoring, and fusing the component sensors using a Bayesian classifier. Empirical results show that **FFM** can detect single and double flux behavior in real time with operationally acceptable probability of detection and false alarm rates. Besides, the empirical results demonstrate that the collected fast flux database can be effectively queried to build automated reports for the security analyst.

❖ In [27], the further improvement was done by authors of this paper, through developing more suitable method for the detection of the Fast Flux Networks and SQL Injection Attacks. The authors proposed using **Machine learning methods** for detection, which provides an accurate means for improving network egress filtering.

❖ In [28], the authors discuss the usage of **SQLMap** in protection of the (SQLI + DNS) Attack. The **SQLMap** has the feature of the DNS Ex filtration and there are many command lines specially designed for DNS prevention and detection. It is compatible with most of the SQL Database versions.

❖ In [29], the authors suggested using a technique of **Crypto-graphical Hash functions** for protection from SQLI + Insufficient Authentication. The hash functions are automatically generated using Hash Algorithms. In this method two extra attributes are added which are hash functions for the username and password field. Where the hash value for each username and password is transferred to the server side for verification.

❖ In [30], the authors proposed using **Noxes** tool for protection against XSS attacks + SQL Injection attacks but **Noxes** fails to prevent the attack completely since the attacker can use HTML Tags instead of Script Tags for an Attack., the authors suggest take care about HTTP request and prevents the modification done on the HTTP header and has the functionality to set cookies.

❖ In [31], the authors proposed using **Ardilla** tool for the detection of the SQLI + XSS attack. It has two modes to check the validity of the attack i.e. strict mode and lenient mode. Ardilla Tool uses Taint Based approaches and static analysis techniques, in this if the preconditions are not met, it will suggest filters and other sanitization methods in order to fulfill the precondition which is the requirement for the detection of a vulnerability.

❖ In [32], the authors developed a **Session Shield**, which is light weight client side protection mechanism, in order to prevent the DNS Hijacking attack.

VI. EVALUATION

As mention in Section IV, SQLIA is categorized into (classical and modern) types, we use them to evaluate the effectiveness of the techniques and tools which are used for detection and prevention of SQLIA (see Section V).

Thus, in this section we follow the same procedure that is followed by [5] to evaluate the impact of techniques (mentioned in Section V-A) against classical types and modern types of SQLIA as demonstrated in the Table1. In Table 1 ,The symbol "✓" is used for technique that can successfully stop all attacks of that type, where the symbol "×" is used for technique that is not able to stop attacks of that type, and the symbol "+" refers to technique that partially stops the attack because of limitations of the underlying approach. As mention in [5], only few of detection and prevention techniques are implemented practically, thereby the results mentioned in Table 1 is based on analytical evaluation rather than empirical result.

Moreover, in Section V-B, we discussed the different techniques and tools that are used in order to detect and prevent modern types of SQLIA, thus, it is worthwhile to evaluate these techniques based on whether that technique is used to detect the attack only or prevent the attack only or it is needed to other methods to complete its action as mention in [8].

In Table 2 we use symbol (✓) for showing that the technique is used for both Detection and Prevention in respect with the modern type of SQL Injections. The letter (D) is used for showing that it is only used for the Detection mechanism. The letter (P) is used for only prevention mechanism. The symbol times (x) is used to depict that techniques or tools does not correspond for the modern SQL Injection (In terms of Prevention and Detection). The symbol (+) depicts the incompleteness which means other method has to be applied in order to achieve complete detection and prevention.

Table 1: Comparison of Detection and Prevention Techniques, mention in Section (V-A), with Respect to SQLIA Types

Type of attack / Techniques	Tautologies	Logically Incorrect Queries	Union Query	Piggy Query	Stored Procedures	Inference	Alternate Encoding	Modern types of SQLIAs
AMNESIA [9]	✓	✓	✓	✓	×	✓	✓	×
[6]	✓	✓	✓	✓	✓	✓	✓	×
SAFELI [10]	×	✓	✓	✓	✓	✓	✓	×
WASP [11]	✓	✓	✓	✓	✓	✓	✓	×
R-WASP [12]	✓	✓	✓	✓	×	✓	✓	×
RT-WASP [1]	✓	✓	✓	✓	✓	✓	✓	×
SecuriFly [14]	+	+	+	+	+	+	+	×
JDBC-Checker [15]	+	+	+	+	+	+	+	×
CANDID [16]	+	+	+	+	+	+	+	×
Swaddler [17]	+	+	+	+	+	+	+	×
DIWeDa [18]	×	×	×	×	×	✓	×	×
Positive Tainting [19]	✓	✓	✓	✓	✓	✓	✓	×
SQL Prevent [20]	✓	✓	✓	✓	✓	✓	✓	×
Automated approaches [21]	✓	✓	✓	✓	×	✓	×	×
SQLIPA [22]	✓	×	×	×	×	×	×	×
PDO[24]	✓	×	✓	×	×	×	×	×

Table 2: Comparison of Detection and Prevention Techniques, mention in Section (V-B), with Respect to Modern Types of SQLIA.

Techniques	**Fast Flux SQLIA**	**SQLI+ XSS**	**SQLI + DNS Hijacking**	**SQLIA using Cross Domain Policies**	**SQLI + DDos**	**SQLI + Insufficient Authentication**
Fast Flux Monitor [26]	**D**	✘	**D**	✘	✘	✘
Machine learning methods [27]	**D**	✓	✘	✘	**D**	✘
SQLMap [28]	✘	**D**	+	✘	**D**	**D**
Crypto-Graphical Hash Function [29]	✘	✘	✘	✘	✘	**P**
Noxes tool [30]	✘	+	✘	**P**	✘	✘
Ardilla tool [31]	✘	**D**	✘	**D**	✘	✘
Session Shield [32]	✘	✓	+	✘	✘	+

VII. CONCLUSION

It is obvious from above discussion and analysis that SQLIAs are one of the most threat to the applications, whether they are web, mobile or desktop applications, that are connected to the database. In this paper we have surveyed the most popular existing attack issues, which is SQLIA. Also we have presented a survey report on classical and modern types of SQLIA, their working methods, and detection and prevention techniques against classical and modern types of that attack. For evaluation, we compare the detection and prevention techniques in terms of their ability to detect the attack, or prevent the attack or partially stop the attack. Regarding the results, the efficiency of some techniques should be improved to overcome the SQLIA.

REFERENCES

[1] N.S. Ali, A. Shibghatullah, “Protection Web Applications using Real-Time Technique to Detect Structured Query Language Injection Attacks”, International Journal of Computer Applications (IJCA), Volume 149, paperNo:6, September 2016.
[2] Goadrich M. and Rogers M., "Smartphone Development: iOS versus Android", Proceedings of the 42nd ACM Technical Symposium on Computer Science Education, Dallas, Texas, USA, PP. 607 612, march 2011.
[3] Meier R., "Professional Android 4 application development", third Edition, John Wiley and Sons, Inc., Canada, 2012.
[4] R.Elmasri, S.B. Navathe, “FUNDAMENTALS OF Database Systems”, sixth edition, Addison-Wesley, United States of America, 2011.
[5] V. Nithya,,R.Regan, J.vijayaraghavan, “ A Survey on SQL Injection attacks, their Detection and Prevention Techniques”, International Journal Of Engineering And Computer Science (IJECS), Volume 2 Issue 4 Page No. 886-905, April, 2013
[6] A. Alazab , A. Khresiat , “ New Strategy for Mitigating of SQL Injection Attack”, International Journal of Computer Applications (IJCA), Volume 154, paper No.11, November 2016.
[7] S. Som, S. Sinha, R. Kataria, “STUDY ON SQL INJECTION ATTACKS: MODE, DETECTION AND PREVENTION”, International Journal of Engineering Applied Sciences and Technology (IJEAST), Vol. 1, Issue 8, ISSN No. 2455-2143, 2016.
[8] Jai Puneet Singh, “Analysis of SQL Injection Detection Techniques”, Theoretical and Applied Informatics (TAAI), Volume. 28, Number. 1-2 , pages 37--55 2016.
[9] W. G. J. Halfond and A. Orso, "Preventing SQL injection attacks using AMNESIA," presented at the Proceedings of the 28th international conference on Software engineering (ICSE), ACM, Shanghai, China, Pages: 795-798, May 20–28, 2006, 2006.
[10] X.Fu, X. Lu, B. Peltsverger, S. Chen, G. Southwestern, K. Qian, and S. Polytechnic, “A Static Analysis Framework For Detecting SQL Injection Vulnerabilities” 31st Annual International Computer Software and Applications Conference(COMPSAC 2007), IEEE, **ISSN:** 0730-3157, pages Number 1–8. , China ,2007.
[11] W. G. J. Halfond, A. Orso, and I. C. Society, “WASP: Protecting Web Applications Using Positive Tainting and Syntax-Aware Evaluation”, **:** IEEE Transactions on Software Engineering, volume. 34, Issue 1, pages. 65–81, 2008.
[12] M. H. A. S. P. Medhane, “R-WASP: Real Time-Web Application SQL Injection Detector and Preventer”, International Journal of Innovative Technology and Exploring Engineering (IJITEE) ISSN: 2278-3075, Volume-2, Issue-5, pages. 327–330April 2013.

[13] S. Manmadhan , Manesh T. , "A METHOD OF DETECTING SQL INJECTION ATTACK TO SECURE WEB APPLICATIONS", International Journal of Distributed and Parallel Systems (IJDPS) ,Volume.3, Issue.6, November 2012.
[14] M. Martin, B. Livshits, and M. S. Lam., "Finding Application Errors and Security Flaws Using PQL: A Program Query Language" ACM SIGPLAN Notices, Volume: 40, Issue: 10 Pages: 365-383, 2005.
[15] 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 (ICSE04) Formal Demos, ACM, ISBN: 0-7695-2163-0, pages 697–698, 2004.
[16] P. Bisht, P. Madhusudan, and V. N. Venkatakrishnan, "CANDID: Dynamic Candidate Evaluations for Automatic Prevention of SQL Injection Attacks", ACM Transaction on information System Security, pages.1–39, 2010.
[17] Macro Cova, Davide Balzarotti." Swaddler: An Approach for the Anomaly-based Detection of State Violations in Web Applications", In Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID), pages: 63–86, (2007)
[18] A. Roichman, E. Gudes, "DIWeDa - Detecting Intrusions in Web Databases". in Proceeding of the 22nd annual IFIP WG 11.3 working conference on Data and Applications Security, Springer, volume. 5094, pages. 313–329, Heidelberg (2008).
[19] William G. Halfond, Alessandro Orso, "Using Positive Tainting and Syntax Aware Evaluation to Counter SQL Injection Attacks", 14th ACM SIGSOFT international symposium on Foundations of software engineering, ACM. pages: 175 – 185, 2006.
[20] P.Grazie., PhD, "SQL Prevent thesis", University of British Columbia (UBC) Vancouver, Canada, 2008.
[21] Mei Junjin, "An Approach for SQL Injection Vulnerability Detection" Proceedings of the 2009 Sixth International Conference on Information Technology: New Generations, IEEE computer society, Las Vegas, Pages 1411-1414, April. 2009.
[22] S. Ali, SK. Shahzad and H. Javed, "SQLIPA: An Authentication Mechanism against SQL Injection", European Journal of Scientific Research, Volume.38, Number.4, pages: 604-611, 2009.
[23] Stephen Thomas, Laurie Williams, "Using Automated Fix Generation to Secure SQL Statements", Proceedings of the Third International Workshop on Software Engineering for Secure Systems (SESS '07), page 9, May 2007.
[24] M. Sendiang, A. Polii, J. Mappadang, "Minimization of SQL Injection in Scheduling Application Development", International Conference on Knowledge Creation and Intelligent Computing (KCIC), IEEE, Indonesia, November 2016.
[25] Android: saving data, Available on :https://developer.android.com/training/basics/data-storage/databases.html [Accessed 29 July 2017].
[26] A. Caglayan, M. Toothaker, D. Drapeau, D. Burke, and G. Eaton, "Real-time detection of fast flux service networks" Conference For Homeland Security, 2009. CATCH '09. Cybersecurity Applications & Technology. IEEE, pages: 285–292, 2009.
[27] E. Stalmans and B. Irwin, "A framework for DNS based detection and mitigation of malware infections on a network", IEEE Information Security South Africa, Johannesburg, pages. 1–8, 2011.
[28] M. Stampar.," Data Retrieval over DNS in SQL Injection Attacks." Available on: http://arxiv.org/abs/1303.3047, 2013.
[29] S. P. Singh, U.N.Tripathi, M. Mishra "Detection and prevention of SQL injection attack using hashing technique", International Journal of Modern Communication Technologies & Research (IJMCTR), volume: 2, Issue 9, September 2014.
[30] E. Kirda, C. Kruegel, G. Vigna, and N. Jovanovic, "Noxes: a client-side solution for mitigating cross-site scripting attacks", in Proceedings of the 2006 ACM symposium on Applied computing. ACM, pages. 330–337, 2006.
[31] A. Kieyzun, P. J. Guo, K. Jayaraman, and M. D. Ernst, "Automatic creation of SQL injection and cross-site scripting attacks," in Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on. IEEE, 2009, pages: 199–209.
[32] N. Nikiforakis, W. Meert, Y. Younan, M. Johns, and W. Joosen, "Session shield: Lightweight protection against session hijacking", International Symposium on Engineering Secure Software and Systems, Springer, volume 6542, pages: 87–100, 2011.