SQL Injection Attack Roadmap and Fusion

Kabir Umar*, Abu Bakar Md Sultan, Hazura Zulzalil, Novia Admodisastro, and Mohd Taufik Abdullah

Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, Serdang, Selangor, Malaysia; kbrumar@yahoo.com

Abstract

With SQL Injection, an attacker can change the intended effect of dynamically generated query in a web Application. This can lead to unauthorized access to the database underlying web application, and harmful transactions on the potentially sensitive information contained in the database. Clear understanding of a problem always assists in finding stronger solution to the problem. In this paper, we conducted an extensive review of several empirical studies on SQL injection attacks and vulnerabilities, with the goal of providing the research community with better insight into possible relationship that exists between different types of SQL Injection Attacks (SQLIAs), and the types of vulnerabilities exploited by each. Consequently, the result of our study is presentation of SQLIAs fusion which shows how different types of SQLIAs lead to one another, and also presentation of step by step SQLIA roadmap. We are very optimistic that our study can help the research community with clearer understanding of SQL Injections, and thus facilitates emergence of stronger solutions to the long standing problem.

Keywords: Attack Intents, Attack Mechanism, Inter-attacks Relationship, Vulnerabilities Exploitation, Web Applications

1. Introduction

Since the beginning of dot com boom in the late 1990s (1997 to 2000), government's organizations, companies and businesses continued to adopt web based applications as a means for delivery of wide range of information and services. Recent developments in web technology such as web 2.0 are rapidly providing more opportunities for internet usage through web applications. Examples of these new opportunities include social networks, enterprise-wide systems, cloud computing, cyber physical objects, etc.

However, as the web applications provide more functionalities, gain more popularity and acceptance, they are often vulnerable to series of attacks from hackers. The intent and impact of these attacks varies from basic information disclosure, remote code execution, memory corruption, denial of service14, cookie theft, session riding14, and total system compromise1. SQL Injection Attack (SQLIA) is one of the most serious security threats facing web applications. SQLIA is an old hacking approach, but still popular among attackers3,4. It was rated as the top in 2011 CWE/SANS top 25 most dangerous software errors3. SQLIAs allow attackers to gain unauthorized and, sometimes, unrestricted access to the databases underlying web application as well as the potentially sensitive information these databases contain13.

Although many SQLIA countermeasures have been proposed by different researchers, the problem of SQLIAs is still posing serious security challenge to web applications, signifying the urgent need for more research towards developing stronger solutions. Understanding a problem always help researchers to develop stronger solution for it. In this paper, we conducted an extensive review of several empirical studies on SQL injection attacks and vulnerabilities, with the goal of providing the research community with better insight into possible relationship that exists between different types of SQLIAs, and the types of vulnerabilities exploited by each.

The result of our study is presentation of SQLIAs fusion which shows how different types of SQLIAs lead to one another, and also presentation of step by step SQLIA
roadmap. We are very optimistic that our study can help the research community with clearer understanding of SQL Injections, and thus facilitating emergence of stronger solutions to the long standing problem. The remaining of this paper is organized as follows. Section 2 presents background on SQLIAs. Within this section, attack intents are discussed in sub Section 2.1, Vulnerabilities exploited by attacks are discussed in sub Section 2.2, while attack mechanisms are discussed in sub section 2.3. Section 3 presents different types of SQLIAs. Section 4 presents SQL injection attack roadmap. Section 5 presents SQL injection attacks fusion. Finally, conclusion is presented in Section 6.

2. Background on SQLIAs

SQLIAs are fast becoming more dangerous and widely spread method of hacking web applications. It is a kind of attack whereby the attacker changes the desired effect of dynamically generated query by inserting additional SQL keywords or operators into the query\(^\text{13}\). Successful SQLIA grants the attacker unauthorized access to the database underlying the web application. The impact of SQLIA ranges from basic information disclosure, remote code execution to total system compromise\(^\text{2,7,10}\).

Different types of SQLIAs exist with differing magnitude and severity of impact. The danger posed by SQLIAs has raised very serious concerns within both industry and research communities, thus, leading to several empirical studies on SQLIA counter measures. In the following sub sections, we present four important attributes of SQLIAs, namely, attack intents, vulnerabilities exploited by attack, and attack mechanism.

2.1 Attack Intents

Attack intents describe possible reasons or motivations that could make an attacker to carry out SQLIA. However, combination of multiple motivations may lead to an attack or sequence of successive attacks. Prominent motivations are described below.

- **Expose Vulnerable Parameters**: The attacker wants to know what weaknesses exist in the web application that can be exploited by SQL injection attack. This information may be used in conducting more serious and targeted attack.

- **Perform Database Finger-print**: The attacker wants to find out the specific type and version of the database that the web application is using.

- **Expose Database Schema**: The attacker wants to discover design related information about the underlying database such as tables’ names, fields’ names, data types, etc. This information may be used in conducting more serious and targeted attack.

- **Extract Data**: The attacker wants to have access to, discover and extract private or confidential information stored in the underlying database.

- **Perform Denial of Service**: The attacker wants to prevent legitimate clients from accessing services provided by the web application. This could be by locking up or dropping database tables, shutting down the underlying database, etc\(^\text{13}\).

- **Database Modification**: The attacker wants to make changes to the contents of the underlying databases, such as edit, add or delete.

- **Remote Code Execution**: The attacker wants to send and execute SQL codes or commands on the underlying database, for example, by using stored procedure or functions.

- **Bypass Authentication**: The attacker wants to have unauthorized access to the web application by avoiding proper authentication mechanism.

- **Total System Compromise**: The attacker wants to have full control over the operation of the system through illegal intrusion.

- **Evade Detection**: The attacker wants to avoid being tracked or detected by the application's protection mechanism.

2.2 Vulnerabilities Exploited by Attack

This refers to the possible software security loopholes or flaws that are associated with web applications and which could be exploited by means of SQLIAs. Commonly known flaws are described below.

- **Insecure Coding Practice**: When the developer is either ignorant of techniques that improve application’s security or fails to implement them properly\(^\text{12}\).

- **Poor Input Data Validation**: Inclusion of user submitted input data into generation of dynamic queries without sufficiently checking the safety and legitimacy of the input data first\(^\text{9}\).

- **Poor Output Sanitization**: Very insufficient or no checking of likely output of dynamic queries
prior to execution of such queries on the database server.

- **Error Message Feedback**: Error messages that are generated by the underlying database server and displayed in the client’s browser can be used by an attacker to detect the type, version or structure of the underlying database.

- **Dynamic Query Generation**: The “input data variables” component of dynamically generated query can be exploited by attacker if no proper input validation or output sanitization is done by the web application.

- **Stored Procedures**: System’s built-in stored procedures are well known to attackers. Also, programmer-built stored procedures may be revealed through some forms of SQLIA in order to “Expose Database Schema”. Consequently, in the absence of proper input validation and output sanitization, an attacker may access and execute both system built-in and programmer-built stored procedures.

- **Generous Privilege**: An ‘admin’ or ‘user’ account with more privileges has access to more database objects and can perform more operations such as SELECT, INSERT, DROP, etc. Where an attacker is able to use such privileged account and bypass authentication, then he/she will have all the privileges associated with the account.

### 2.3 Attack Mechanism

This refers to the means or channels used by hackers to perform SQLIAs. Prominent attack mechanisms are described below.

- **User Input Injection**: This is a situation where an attacker submits malicious SQL commands or string as input data via form’s data entry fields. Upon submission, the malicious input data (contained in the http GET or POST request) will be accessed in a way other variables are accessed. If the malicious input data is eventually used in generation and execution of dynamic query, then that will result in SQL injection.

- **Cookies Tempering**: Some web applications generate and store cookies on the client’s machine for tracking client’s state information. When the client returns to the site, the web application uses the content of the cookie to restore client’s state information. Where the web application uses cookie content to build dynamic queries, an attacker can temper with or change the content of such cookie and inject malicious string\(^13\).

- **Through Server Variables**: Server variables, such as HTTP, network headers and environment variables, are used by web applications in variety of ways including logging usage statistics and identification of browsing trends. When these variables are logged to the database without effective sanitization, an attacker can forge the values placed in HTTP and network headers with malicious string. When the query to log these server variables is issued to the database then the attack is triggered. This kind of attack is known as second order SQLIA.

### 3. Types of SQLIAs

SQLIAs are classified based on the way an attack is carried out by an attacker, the intent of the attacker as well as the vulnerabilities exploited by the attack. There are seven well known types of SQLIAs. For each type of SQLIA presented in this study, we discuss the following items.

- Description of the attack.
- Attack intent.
- Vulnerabilities exploited by the attack.
- How the attack is carried out.

The seven types of SQLIAs discussed below includes: 1. Tautologies, 2. Logically incorrect queries, 3. Union queries, 4. Piggy-backed queries, 5. Stored procedures, 6. Inference, and 7. Alternate encodings.

### 3.1 Tautology

- **Description**: This attack generally injects code that causes tautology in one or more conditional statements of SQL query, thereby, making the statements to ALWAYS evaluate to true.

- **Attack Intent**: Extracting data, Bypassing authentication, Exposing vulnerable parameters.

- **Vulnerabilities Exploited**: Poor input validation, Dynamic SQL query generation.

- **How Attack is Carried Out**: This attack is mostly carried out through user input data supplied via an online form and submitted through HTTP GET or POST request. The injected code must be syntactically correct and be able to evaluate to true\(^5,7,13\).
3.2 Logically Incorrect Queries

- **Description:** This attack reveals important information about the type, version and structure of the database underlying a web application. The information is revealed through display of error messages in the client's browser. This type of attack is usually considered as a preliminary information gathering step towards more severe and target-specific attacks.
- **Attack Intent:** Extracting data, Performing database fingerprint, Exposing vulnerable parameters.
- **Vulnerabilities Exploited:** Insecure coding practice, Error Message Feedback, Poor input validation, Dynamic SQL query generation;
- **How Attack is carried out:** An attacker carries out this attack by injecting malicious SQL string that cause a syntax error, type conversion error, or logical error. The database server responds by generating an appropriate error message displayed in the client's browser\(^5,7,12,13\).

3.3 Union Queries

- **Description:** This attack is used to change the dataset returned by a given query. The attacker can trick the application into returning data from a table different from the one intended by the original query. The returned dataset is the union of the results of the original query and the injected query;
- **Attack Intent:** Extracting data, Bypassing Authentication.
- **Vulnerabilities Exploited:** Insecure coding practice, Poor input validation, Poor output sanitization, Dynamic SQL query generation;
- **How Attack is carried out:** This attack is carried out by injecting a SQL query string that insert SELECT query into dynamically generated query using the `UNION` keyword. The injected SQL string is generally of the form: `UNION SELECT <rest of injected query>`\(^5,6,11\).

3.4 Piggy-Backed Queries

- **Description:** This is an attack where the attacker injects additional query into the original query without modifying the original developer intended query. This type of attack can allows attacker to insert virtually any type of SQL commands, including stored procedures. The injected query is executed along with the original query.
- **Attack Intent:** Extracting data, Database modification, performing denial of service, remote code execution.
- **Vulnerabilities Exploited:** Insecure coding practice, Poor input validation, Dynamic SQL query generation
- **How Attack is carried out:** This attack is carried out by injecting additional query as an attachment to the original query, without modifying the original developer intended query. The new and distinct added query is "piggy backed" on the original query. This causes the database server to receive and execute multiple queries, though, only the first query is developer intended while the subsequent ones are injected queries. The original developer intended query is executed along with the added queries. This type of attack often exploits database configuration which allows multiple statements to be contained in a single SQL string\(^5,11,13\).

3.5 Stored Procedures

- **Description:** If an attacker is able to determine which back-end database server is in use (through other types of attacks), then he/she can design attack to execute built-in stored procedures that are provided by that specific database server. In addition, an attacker may discover and execute developer-built stored procedure as well.
- **Attack Intent:** Performing denial of service, performing privilege escalation, Remote code execution.
- **Vulnerabilities Exploited:** Stored procedures, Poor input validation, Poor output sanitization, Insecure coding practice.
- **How Attack is Carried out:** This attack is carried out by injecting malicious string into stored procedures, and triggering their execution with the injected malicious input\(^5,7,13\).

3.6 Inference

- **Description:** This attack modifies developer intended query into the form of action which
is executed based on the answer to a true/false question. The attacker carefully observes response and behaviour of the website when subjected to different attacks. From the application's responses, an attacker can infer certain vulnerable parameters, database schema information, and so on. There are two well-known variations of this type of attack:

- **Blind Injection:** The attacker asks the server true/false questions, and infer the answer by observing the behavior of the website after an attack. If the website continues to function normally after an attack, it means the injected malicious string evaluates to true. However, if the observed behavior of the website changes after an attack, it means the injected malicious string evaluates to false.

- **Timing Attack:** The attacker structures the injected malicious string in the form of an if...then...structure. The malicious string is crafted to contain SQL construct that takes a known amount of time to execute such as WAITFOR keyword along different if...then... branches. By measuring application's response time after different attacks, the attacker can infer which branch was taken in his injected string. Thus, the attacker can infer some information as to why such branch of his injected string was taken.

- **Attack Intent:** Evade detection, Remote code execution.

- **Vulnerabilities Exploited:** Poor input validation, Insecure coding practice, Character Escaping, Generous privilege.

- **How Attack is Carried out:** The attacker converts the injection string into an alternate encoding system and then injects into a vulnerable field. Because the injected string is in alternate encoded format, it would be considered harmless by most of the application's defensive mechanism, and would be allowed to execute on the database server. Unfortunately, at the database level prior to execution, the injected string is interpreted into normal format, and eventually results in malicious SQL injection\(^3,6,13\).

### 4. SQL Injection Attack Roadmap

We carefully analysed how each of the above types of SQLIA is carried out. Consequently, we propose the following attack roadmap (Figure 1.) which describes the sequence of activities done by an attacker from planning to launch of attack. Each node on the attack roadmap corresponds to some activity performed by the attacker in the order shown in Figure 1. The activities at all steps are described below.

![SQL Injection Attack Roadmap](image)

**Legend**
- Set attack target
- Stages of attack progress
- Set attack intent
- Attacker's activity
- Find vulnerable fields
- Loop backs
- Select attack mechanism
- Craft attack string
- Launch the attack

**Figure 1. SQL injection attack roadmap.**

- **Set Attack Target:** The attacker defines target of the attack. He/she defines which particular web application or web applications to attack.

- **Set Attack Objective:** The attacker defines the main intents or motivations for conducting the attack. For example, intent could be to cause

### 3.7 Alternate Encodings

- **Description:** This type of attack is simply a technique that allows attacker to evade detection, and exploits vulnerabilities that might not otherwise be exploitable. The injected malicious string is written in alternate methods of encoding such as hexadecimal, ASCII, and unicode character encoding.

- **Attack Intent:** Evade detection, Remote code execution.

- **Vulnerabilities Exploited:** Poor input validation, Insecure coding practice, Character Escaping, Generous privilege.

- **How Attack is Carried out:** The attacker converts the injection string into an alternate encoding system and then injects into a vulnerable field. Because the injected string is in alternate encoded format, it would be considered harmless by most of the application's defensive mechanism, and would be allowed to execute on the database server. Unfortunately, at the database level prior to execution, the injected string is interpreted into normal format, and eventually results in malicious SQL injection\(^3,6,13\).
denial of service in the target web application or to extract private and confidential information.

- **Find Vulnerable Fields**: The attacker finds available exploitable vulnerabilities in the target web application. He/she experiments on several data entry fields and then observes the application’s responses or error messages. Eventually, the attacker may have a clue as to what vulnerabilities exist for exploitation.

- **Select Attack Mechanism**: Having defined attack intent and got clue about exploitable vulnerabilities, the attacker defines appropriate mechanism by which to conduct the attack. For example, if the attack intent is to extract customers’ information, and the attacker discovered that the web application uses contents of cookie to generate and execute dynamic query, then the attacker may choose “Cookie Tempering” as suitable attack mechanism.

- **Craft Attack String**: The attacker crafts the actual attack string in form of SQL string, SQL commands or alternate coding. The crafted attack string is in a form that can be injected through user input, cookie tempering, etc.

- **Launch the Attack**: The attacker launches the real attack. Successful attack can lead to devastating consequences to the target web application. However, when attack fails, the attacker loops back to an appropriate previous step of the roadmap to consider different alternatives and try again.

The figure above shows diagrammatic representation of the SQLIA roadmap. The aim of the roadmap is to provide researchers with clear view of the overall activities done by hackers from planning to launch of SQLIAs.

### 5. SQL Injection Attacks Fusion

The relationship that exists between different types of SQLIAs has not been investigated in the current literature. Consequently, we conducted review of several empirical studies on SQL injection attacks and vulnerabilities in order to find out the possible existence of such inter-attacks relationships. We assessed how different types of SQLIAs are carried out, the vulnerabilities exploited by each, and the motivations that can trigger each type of attacks.

Our study leads us to presentation of SQL Injection Attacks fusion. The SQLIAs fusion is a lattice that represents relationships among different types of attacks. It shows which attack can embed another, and what cumulative intents are achievable by each attack. The SQLIAs fusion is shown in Figure 2. For example, Tautology attack can be embedded within (or fused into) Stored procedure attack. This implies that an application that is vulnerable to stored procedure attack can as well be exploited using Tautology attack. In general, the attacks fusion implies that if attack “A” can be embedded within attack “B”, then any application exploited with attack “B” can as well be exploited with attack “A”. In addition, all attack intents achievable with “A” can as well be achieved with “B”. It is worth noting that, the SQL Injection attacks fusion relationship is not symmetric, hence, the reverse does not apply.

**Figure 2.** Relationship between different types of SQLIAs.
The fusion relationship is depicted by the Lattice diagram of Figure 2. In the diagram, an attack is represented by oval, the set of Intents achievable with the attack are represented by linked cells, and filled circle represents lattice node. A node connects an attack to the set of achievable Intents. Nodes are connected to show fusion relationships. From the diagram, an attack at a lower node can be fused into any attack at the next upper node. For instance, Tautologies attack, which is the lowest node, can be fused into Stored procedure attack as well as Union query attack. The effect of this is, first, Tautology attack can be carried out within either Stored procedure attack or Union query attack, and second, Tautology attack intents, such as “bypassing authentication”, can be achieved by carrying out Stored procedure attack or Union query attack. From technical perspective, this means for developers to effectively prevent stored procedure attack, they must take both Tautology attack and stored procedure attack into consideration.

The diagram is traversed bottom-up (i.e., from the bottom to the top). Beginning with Tautology attack (the lowest node) as you move up to the next node, you can fuse Tautology attack into the attack at the next upper node and gain attack intents advantage. For example, you can fuse Tautology into Stored produce attack and achieve the attack intents at both (connected) nodes. Thus, if you fuse Tautology attack into stored procedure attack, you can achieve: ExD|ByA| VuLP + DoS|RCod|PrvE

The fusion diagram is designed to help researchers to have a deeper insight into, and be able to further explore the relationship that exist between different types of SQL Injection Attacks. We are optimistic that this will facilitates development of more secure defensive mechanism for protecting web applications against the long standing and challenging problem of SQLIAs. Note that the SQL Injection Attacks fusion lattice diagram is not symmetric.

6. Conclusion
The possible relationship between different types of SQL Injection attacks has not been investigated in the current literature. In this paper we conducted extensive review of several empirical studies that addressed SQL injection attacks and vulnerabilities. Our study leads us to presentation of SQLIAs fusion and roadmap. SQLIAs fusion depicts inter-attacks relationships using a non symmetric lattice traversed bottom-up. The roadmap presents sequence of activities conducted by attackers from planning to launch of an attack. Clear understanding of a problem always assists in finding stronger solution to the problem. Therefore, we are very optimistic that our study can help the research community with clearer understanding of SQL Injections, and thus facilitating emergence of stronger solutions to the long standing problem.

7. Acknowledgement
We acknowledge that this research received support from the Fundamental Research Grant Scheme FRGS/1/2015/ICT01/UPM/02/12 awarded by Malaysian Ministry of Education to the Faculty of Computer Science and Information Technology at Universiti Putra Malaysia.

8. References
1. 2011 CWE-SANS Top 25 most dangerous software errors; 2015. Available from: http://cwe.mitre.org/top25/
2. Watson D. Web application attacks. Network Security. 2007; (10):10–14. DOI: 10.1016/S1353-4858(07)70094-6.
3. Focardi R, Luccio FL, Squarcina M. Fast SQL blind injections in high latency networks. IEEE Proceedings of the 1st AESS European Conference on Satellite Telecommunications, ESTEL’12; 2012 Oct 2–5. Rome, Italy. p. 1–6. DOI: 10.1109/ESTEL.2012.6400112.
4. Garg A, Singh S. A review on web application security vulnerabilities. Advanced Research in Computer Science and Software Engineering. 2013; 3(1):222–6. Available from: http://www.ijarcse.com/docs/papers/Volume_3/1_January2013/V3I1-0196.pdf
5. Balasundaram I, Ramaraj E. An efficient technique for detection and prevention of SQL injection attack using ASCII based string matching. Procedia Engineering. 2012; 30(2011):183–90. DOI: 10.1016/j.proeng.2012.01.850
6. Lee I, Jeong S, Yeo S, Moon J. A novel method for SQL injection attack detection based on removing SQL query attribute values. Mathematical and Computer Modelling. 2012; 55(1–2):58–68. DOI: 10.1016/j.mcm.2011.01.050.
7. Abawajy J. SQLIA detection and prevention approach for RFID systems. Systems and Software. 2013; 86(3):751–8. DOI: 10.1016/j.sss.2012.11.022.
8. Kumar P, Pateriya RK. A survey on SQL injection attacks, detection and prevention techniques. IEEE Proceedings of the 3rd International Conference on Computing Communication and Networking Technologies, ICCCNT ’12. 2012 Jul 26–28. p. 1–5. DOI: 10.1109/ICCCNT.2012.6396096.
9. Rahul J, Pankaj S. A survey on web application vulnerabilities (SQLIA, XSS) exploitation and security engine for sql injection. Proceedings of the International Conference on Communication Systems and Network Technologies; 2012 May. p. 453–8. DOI: 10.1109/CSNT.2012.104.

10. Simone C, Diane G. SQL injection attacks with the AMPA suite. Electronic Security and Digital Forensics. 2013; (5):2. DOI: 10.1504/IJESDF.2013.055051.

11. Shahriar H, Zulkernine M. Information-theoretic detection of SQL injection attacks. IEEE Proceedings of the 14th International Symposium on High-Assurance Systems Engineering; 2012 Oct. p. 40–7. DOI: 10.1109/HASE.2012.31.

12. Shar LK, Tan HBK. Defeating SQL injection. Computer. 2013; 46(3):6265060):69–77. DOI: 10.1109/MC.2012.283.

13. Halford W, Viegas J, Orso A. A classification of SQL injection attacks and countermeasures. IEEE Proceedings of the International Symposium on Secure Software Engineering, ISSSE’06; Available from: http://www.cc.gatech.edu/fac/Alex.Orso/papers/halfond.viegas.orso.ISSSE06.pdf

14. Web application security; 2015. Available from: http://en.wikipedia.org/wiki/Web_application_security