A second-order SQL injection detection method

Chen Ping
PLA Army Engineering University, Nanjing 210007, China
chenpin0361@sina.com

Abstract—**Second-order SQL injection is a serious threat to Web application and it is more difficult to detect than first-order SQL injection. The attack payload of second-order SQL injection is from untrusted user input and stored in database or file system, the SQL statement submitted by web application is usually dynamically assembled by a trusted constant string in the program and untrusted user input, and the DBMS in unable to distinguish the trusted and untrusted part of a SQL statement. The paper presents a method of detecting second-order SQL injection attacks based on ISR（Instruction Set Randomization）. The method randomizes the trusted SQL keywords contained in Web applications to dynamically build new SQL instruction sets, and add a proxy server before DBMS, the proxy detects whether the received SQL instruction contains standard SQL keywords to find attack behavior. Experimental results show that this system can effectively detect second-order SQL injection attack and has low processing cost.**

Keywords—Second-order SQL injection , ISR（Instruction Set Randomization）, Web application, proxy server

I. INTRODUCTION

With the rapid development of Web technology, Web applications are facing more and more security problems. According to the data released by the open Web application security project (OWASP), the SQL injection vulnerability is the primary vulnerability to Web security [1]. In the past few years, many experts and scholars have done a lot of research on SQL injection vulnerabilities, and proposed many injection methods and defense technologies, but most of these studies are limited to first-order SQL injection. Second-order SQL injection is developed on the basis of the traditional first-order SQL injection technology. It is more difficult to discover, but has the same security threat to Web applications as the first-order SQL injection. At present, there is little research on second-order SQL injection technology. YAN[2] analyze the principle of T-order SQL injection, and propose a method of combining dynamic detection with static detection to detect second-order SQL injection vulnerabilities. Tian Yujie [3] put forward a kind of second-order SQL injection attack defense model based on improved parameterization, including input filter module, index replacement module, syntax comparison module and parametric replacement module. However, these methods are not ideal for the detection of second-order SQL injection vulnerabilities.

The paper analyzes the principle of second-order SQL injection, and presents a method of detecting second-order SQL injection attacks based on ISR（Instruction Set Randomization）. The experimental results show that the proposed method can detect second-order SQL injection accurately and effectively.

II. THE PRINCIPLE OF SECOND-ORDER SQL INJECTION

The first-order SQL injection loads attack payload directly to the SQL query, but the attack process of second-order SQL injection is divided into 2 stages[4], namely the stage of storing attack payload to database or file system and the stage of building the SQL query statement with the attack payload from database or file system. For example, the common function of user registration and user information modification in Web application can carry out an second-order SQL injection attack, the user registration function loads the attack payload to the database, and then the user information modification function extract the attack payload from database to construct a SQL database query, which will cause a second-order SQL injection . The key codes of user registration function are as follows:

```
$link=new mysqli(“localhost”, “root”, “root”) ;

$query=“INSERT INTO Users VALUES(‘?’,’?’,’?’) ” ;

$cmd=$1ink->prepare($qurey) ;

$cmd->bind_param(“sss”,$usemame,$password,$email) ;

$cmd->execute（）
```

The above codes use precompiled statements to insert a user record to the database instead of dynamicly constructing SQL query. When DBMS processes a precompiled statement, it first compiles the SQL command by placeholders instead of the user input, and then enters the user input to execute. Since the compilation has completed, even if the user input contains attack load, it will not be parsed by the database, this is a method of defense against the first-order SQL injection. For example, now insert a user record with the email field “123’ WHERE 1= (updatexml (1, concat (0x5e24 (select password from admin limit 0x5e24, 1, 1)))); -- ”,because the user registration function uses precompilation statements to insert a user record, the attack payload will be treated as general data and stored into the database.

However, the attack payload stored in the database will be reused when modifying the user’s information. The user data modification function of Web application will verify the username and password of the user before changing the password, and extract the corresponding user data from database to memory, the SQL instructions of verification and extraction of user data list as follows:

```
$query=“SELECT * FROM Users WHERE username=‘? ’ AND password=‘? ’ ” ;
```

$result = mysql_query($query)

Web application authenticates user by verifying whether the returned result set is empty, and saves the result set in memory. Web application uses the result set in memory to construct SQL statement for modifying user's passwords, the key code is shown as below.

$result=mysql_query($query) ;

$row=mysql_fetch_array($result) ;

$usemame=$row["usemame"] :

$email=$row["email"] ;

$password=$_POST["newpwd"] ;

$query="UPDATE Users SET username='$username' , password='$password',email='$emal' WHERE usemame='$username' ; " ;

$rs=mysql_query($query)

The above code uses the username and email in memory and the new input password of the user to construct the UPDATE instruction. At this point, the attack payload stored in the email field in database is dynamically constructed the SQL query, the new constructed SQL instructions is shown below.

UPDATE Users SET username='XXX',password = 'XXX', email = '123' WHERE l=(updatexml(1 , concat(0x5e24 , (select password from admin limit 1) , 0x5e24) , 1)) ; --'WHERE username='XXX'

When the DBMS executes the above SQL instruction, a database error is generated because the XPATH string received by the updatexml () function has a syntax error, the error message is shown below:

XPATH syntax error:'$admin123$'

From the error prompt message, the attacker can know the password of user admin is admin123. A second - order SQL injection attack is successfully implemented.

From the above example, we can see that the second-order SQL injection is divided into 2 stages: storage and trigger. In the storage phase, the attack payload is wrote to the user's input and send to the Web application, because Web application has deployed the defense measure, when the INSERT/UPDATE statement is executed by DBMS, the attack payload will be stored in the database. In the trigger phase, Web application will construct a new SQL instruction with the attack payload stored in the database. When the DBMS execute the newly constructed SQL instruction of the Web application, the attack payload is parsed and executed, and a second-order SQL injection is implemented.

The attack payload of the second-order SQL injection can be stored in the file system as well as being stored in the database, which will be analyzed in subsequent chapters.

III. DESIGN AND IMPLEMENTATION OF SECOND-ORDER SQL INJECTION PREVENTION SYSTEM

In the second-order SQL injection, the attack payload stored in database is non-credible data which malicious user input, while the SQL fragment in Web applications is credible, but the existing DBMS cannot distinguish the trusted and untrusted part of a SQL statement. The second-order SQL injection defense mechanism proposed in this paper randomizes the SQL keywords in trusted constant string in Web application, which will create dynamically new sets of SQL keywords, and can distinguish from standard SQL keywords in attack payload which attacker inject and stored in database or file system. We can find the SQL injection attack by detecting whether the SQL statement includes the standard keywords.

In the second-order SQL injection example above, the raw, non-randomized statement contained in the Web application is:

$query="UPDATE Users SET username='$username' , password='$password',email='$emal'WHERE usemame='$username' ; "

The detection method of second-order SQL injection random 3 keywords in the above SQL statement: UPDATE, SET, FROM, for example, if the current random number is 785,the SQL statement after randomization is:

$query="UPDATE785 Users SET785 username='$username',password='$password',email='$emal' WHERE785 usemame='$username' ; "

Then attack payload stored in the email field of database is extract to construct the SQL statement. The full SQL statement is:

UPDATE785 Users SET785 username='XXX',password = 'XXX',email ='123' WHERE l=(updatexml(1 , concat(0x5e24 , (select password from admin limit 1) , 0x5e24), 1)) ; --' WHERE785 username='XXX'

The UPDATE statement contains the standard keywords "select" and "WHERE", these standard keywords come from the malicious user, the second-order SQL injection defense mechanism proposed in this paper will detect the attack.

The second-order SQL injection defense system is composed of randomization module, attack detection module and de-randomization module. Randomization modules append a random integer to SQL keywords in Web application, which creates dynamically SQL keyword sets, the attacker don't know the new SQL keyword sets, so standard SQL which attacker injects can be detected, but at the same time, DBMS can't recognize the new keywords, too, a solution to solve the problem is to modify the database interpreter, but this method is very complex. We adopt the method of building a proxy server between Web server and database server, as shown in Fig. 1, the proxy server has two main functions, one is to detect SQL injection attacks, the other is to de-randomize the harmless instructions to standard SQL statement and forward it to DBMS.

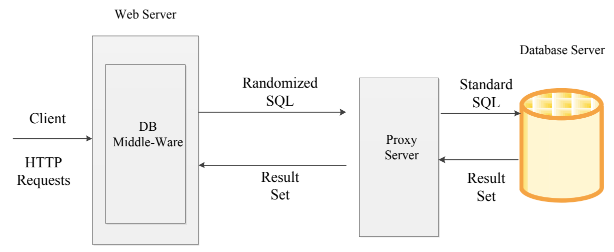


Fig. 1 SQL injection detect system

A. *Keywords randomization*

The standard SQL statement can be described with regular expression, our method use regular expression to locate accurately the trusted constant strings containing SQL statement in the Web application, then randomize the keywords in SQL statement with a random integer.

The keywords for each SQL statement is shown in Table 1, each type of SQL statement has a primary keyword which is formatted by bold font. Due to the WHERE expression in Insert, Delete, Update statement can contain sub queries, so these types of SQL statements should involve the query keywords in addition to its own keywords.

Table I SQL keywords

TYPE	KEYWORDS
Query	**SELECT**、FROM、WHERE、ORDER BY、GROUP BY、HAVING、UNION、OR、AND
Insert	**INSERT**、INTO、VALUES、the query keywords
Delete	**DELETE**、FROM、WHERE、the query keywords
Update	**UPDATE**、SET、WHERE、the query keywords
Table operator	**CREATE** TABLE、**ALTER** TABLE、**DROP** TABLE

The step of keyword randomization is described in Algorithm 1.

Algorithm 1:SQL keyword randomization

Input: Web application file , share key K

Output: the file with randomized SQL keywords

Step:

1. use regular expressions to locate the SQL constant strings in the file;

2. produce a 3-digit random number r, and use the shared key K to encrypt r, and get R, R= F (r, K);

3.determine the type of SQL operation, locate the primary keyword M, randomize the main keyword M as: r+M+R, where + is the string connector;

4. randomize other keyword S in SQL statement as: S+R;

5. complete the randomization of all SQL constant strings according to the above steps.

The system first generates a 3-digit random number r, calculates R= F (r, K), K is the key shared between the randomization middleware and the proxy server, and F is the encryption function. In order to prevent an attacker from impersonating, number r is placed in front of the primary keyword, while number R is placed to the end of the primary key, for example, the primary keyword SELECT is changed to the form r+SELECT+R after randomization ,when proxy server receives a SQL statement ,it will separate the random number r and R from the primary key, calculate F (r, K) with shared key K, compare the result of F (r, K) with R, if the two are not equal, it will conclude that the command is sent by attacker.

B. *Construction of the proxy server*

In figure 1, the proxy server has two main functions, one is to detect SQL injection attacks, the other is to de-randomize the harmless instructions to standard SQL statement and forward it to the DBMS.As the most popular relational database management system, MySQL has become the preferred database system for Web application development. This paper takes MySQL as an example to introduce the construction of proxy server.

MySQL has the characteristics of simple, efficient and reliable, one of the most important applications is to construct the database cluster system, which usually use middleware MySQL Proxy to implement the function of load balancing[5]. MySQL Proxy is deployed between Web server and MySQL database server, and the embedding Lua[6] script has the ability to monitor and modify the communication between client and server. when MySQL proxy works, it first acts as a server, receives instruction from web server (as a client), analysizes and processes them according to the configuration, then acts as a client, sends the instruction to the backend database, and finally accepts the results from the database, transmits them to the web server. MySQL Proxy has the characteristics of convenient deployment, easy to operate, and the expansion of the use of Lua language is easy to learn and use, In this paper, we modify MySQL Proxy to meet our need.

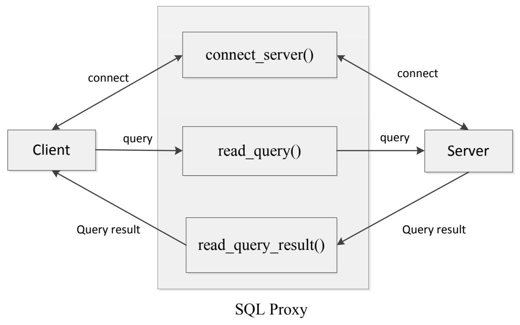


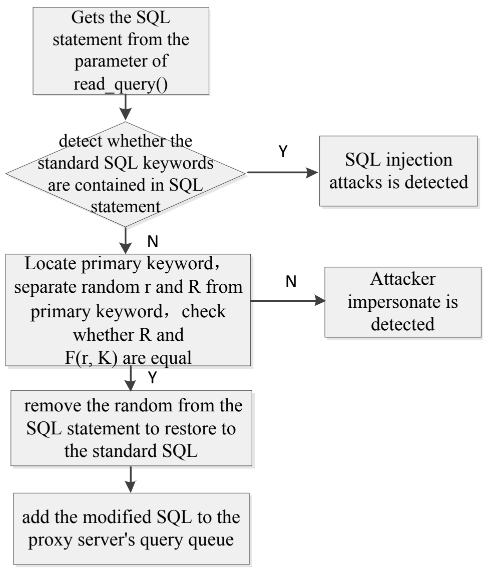
Fig. 2 core functions of MySQL Proxy

MySQL proxy's core functions are connect_server()、read_query()and read_query_result(),shown in Fig.2

connect_server() transmits the connection request, read_query() transmits the SQL statement, read_query_result() transmits the query results. So in order to realize the detection of SQL injection and derandomization of SQL statement, we should rewrite the read_query(). In MySQL proxy, we can modify the startup configuration file mysql-proxy.cnf to rewrite read_query(),we should add the following configuration items :

```
proxy-address = 192.168.1.1:4040
proxy-lua-script = E:/ mysql-proxy/ sqlidetect.lua
proxy-backend-addresses=192.168.1.1:3306
```

Parameter proxy-address specifies the IP and port of the proxy server; parameter proxy-backend-addresses specifies the IP and port of the MySQL database; parameter proxy-lua-script specifies the Lua script to process the SQL command, in the script file, we can rewrite read_query () function, adding the function of SQL injection attack detection and the randomization, the main process is shown in Figure 3:



Gets the SQL statement from the parameter of read_query()
detect whether the standard SQL keywords are contained in SQL statement
Y
SQL injection attacks is detected
N
Locate primary keyword，separate random r and R from primary keyword，check whether R and F(r, K) are equal
N
Attacker impersonate is detected
Y
remove the random from the SQL statement to restore to the standard SQL
add the modified SQL to the proxy server's query queue

Fig. 3 flow chart of attack detecting and randomization

Get the SQL command from parameter packet, detect whether it contains standard keyword, if it contains standard SQL keyword means that the database instruction contains the untrusted SQL fragment which user input, namely there is an SQL injection attack. To prevent attackers from counterfeiting, detection whether two random numbers attached primary key meets a predetermined relationship, finally de-randomize the harmless SQL statement to standard SQL implemented and forward it to the background MySQL.

IV. FUNCTIONAL AND PERFORMANCE TEST OF THE PROTOTYPE SYSTEM

In order to verify the detection ability of the proposed SQL injection detection method for real Web applications, we build experimental environments to evaluate its function and performance respectively.

A. *functional test*

Function test uses webchess、schoolmate、Sql-labs as test object, experimental results show that the proposed SQL injection detection method in this paper can effectively detect second-order SQL injection attack. We take Sqli-labs [7] as an example to introduce the testing process. Sqli-labs is a SQL injection learning software, Sqli-Less24 Web application has second-order SQL injection vulnerability and the attack payload is stored in file system. The application provides new user registration and password modification function, the input user information when registration is escaped by the function mysql_escape_string(), so there isn't first-order SQL injection vulnerability, after successful registration, the user name is stored in $_SESSION, the code is shown as below:

$_SESSION["username"] = $username;

Session is a typical form of storing attack load in file system. The session mechanism is used to solve the problem of recording the status of browse in some scenarios, usually the session initialization code generates a unique session_id , the client will send the session_id in the next request as the identity certificate of the client, the server will retain credential in a corresponding file, storing and accessing session data is through the global variable $_SESSION. When the user want to change the password, the attack payload stored in $_SESSION will be extracted as the query condition:

```
$username= $_SESSION["username"];
$curr_pass= mysql_real_escape_string($_POST['current_password']);
$pass= mysql_real_escape_string($_POST['password']);
$re_pass= mysql_real_escape_string($_POST['re_password']);
if($pass==$re_pass)
{
$sql = "UPDATE users SET PASSWORD='$pass' WHERE username='$username' AND password='$curr_pass' ";
    $res = mysql_query($sql)
}
```

When an attacker registers a username as "' or 1=1--", the username is stored in the $_SESSION and when the user wants to change the password,the UPDATE statement is:

UPDATE users SET PASSWORD='XXX' WHERE username=''or 1=1—' AND password='$curr_pass'

In this statement, the 1=1 in the WHERE condition is always true, which will lead to change the passwords of all users to the password designated by the attacker.

Using the second-order SQL detection mechanism proposed in this paper, the trusted SQL keywords in Web is ransomed, for example, if the random number is 378, the UPDATE statement is:

UPDATE378 users SET378 PASSWORD='XXX' WHERE378 username='' or 1=1--' AND378 password='$curr_pass'

The proxy server will detect that there is a standard SQL keyword "or" in the statement, that is ,a second-order SQL injection is detected.

B. *performance test*

In the performance experiment of the prototype system, we also use webchess、schoolmate、Sql-labs which exist second-order SQL injection vulnerability as the test object, these Web applications are build with PHP and MySQL and applied the SQL injection prevention method proposed in this paper，Table 2 lists the page response time of non-protect system and protected system ,we can see that the defense system only increase by about 1s of additional processing time, which is in the scope of user can accept .

Table 2 The page response time of non-protect system and protected system

Web application	page response time (no protect)	page response time(protect)	Overhead
webchess	268ms	289ms	7.8%
Sql-labs	12ms	13ms	8%
schoolmate	234ms	252ms	7.6%

V. CONCLUSIONS

In this paper, we propose a method of preventing second-order SQL injection attacks and build a prototype system based on this strategy. Experimental results show that this system has a good effect on preventing SQL injection and low running cost. The system does not rely on any database or server platform and is easy to transplant.

REFERENCES

[1] OSWAP . Category : OWASP top ten project [EB/OL] . http://www.owasp.org/index.php/Category : OWASP_Top_Ten_Project.

[2] YAN L，LI X H，FENG R T，et al．Detection method of the second-order SQL injection in Web applications[J]．Lecture Notes in Computer Science，2014，8332：154-165．.

[3] TIAN Y J,ZHAO Z M，ZHANG H C，et al．Second-order SOL injection attack defense model[J]．Netinfo Security,2014,(11)：70-73．

[4] DAHSE J，HOLZ T．Static detection of second-order vulnerabilities in Web applications[A]．Preceedings of the 23rd USENIX Conference On Security Symposium(USENIX [C]．2014．989-1003．

[5] ZHU X F. Research and analysis of database cluster middleware MySQL Proxy [D].Wuhan University of Technology,2011. 8.

[6] WANG Y Z. Lua-A Powerful Scripting Language[J]. Science Plaza,2010,Vol.1:205-207. 9.

[7] Sqli-labs Project[EB/OL].https://github.com/Audi-1/sqli-labs.