The research of an AOP-based approach to the detection and defense of SQL injection attack

Wang Qing[1,a] and Chengwan He[2]

[1]School of Computer Science and Engineering. Wuhan Institute of Technology
[2]Hubei Province Key Laboratory of Intelligent Robot Wuhan, China

Abstract. As the availability of web application services grows, we are witnessing an increase in the number and sophistication of attacks that target them. The SQL injection attack has been the most dangerous way of web-based attacks. In this paper, according to the characteristics of the SQLIAs, we presented a new method for detecting and preventing SQL injection attacks by using AOP. On the one hand, we solve these SQLIAs which have attack characteristics by defining aspect and pointcut, then doing some validations in the function of *before()*. On the other hand, we use a model-based way for other attacks, which uses the program analysis technique to automatically build a model of legitimate SQL queries, and the model is compared with the SQL queries obtained dynamically by AOP. We illustrate the method through a case study- a simple user login page. The results show the effectiveness of our approach.
Keywords: SQL injection attack; AOP; attack characteristic; logical structure.

1 Introduction

According to assessment results of Internet Application safety risk presented in OWASP (Open We Application Security Project)[1], SQL injection attacks [2] ranked first in the top 10. SQL Injection is such a vulnerability : if an attacker affects the SQL query which will be transmitted to database, that will result in injection attack. Meanwhile, the attacker modifies SQL syntax and function by constructing the content passed to the database. The harmness of SQL injection vulnerabilities [3] are not only for Web applications, but also for those codes that obtained input from untrusted data. In the past, the main target of most typical SQL injections [4] was server database, however, according to the specification of current HTML 5, the attacker can also use the same way to access the client database to steal data by executing JavaScript or other code.

AOP (aspect-oriented programming) provides the technique which encapsulates the crosscutting concern as the aspect [5]. For static AOP, the section in the form of byte code compiles into the target file directly during compiling. For dynamic AOP, after the target class loaded, weaved the section into proxy class that generated dynamically for interface during runtime.

In this paper, we proposed an AOP-based approach. For those usual attacks, we solved them by defining aspect and pointcut. For others, we combined static analysis with dynamic monitoring, and weaved the monitoring code into source code by using AOP technology during dynamic monitoring.

[a] Corresponding author： qingwangqw@163.com

2 Related research

In view of the SQLIAs, many scholars have put forward some technology solutions in recent years.

In 2003, Yao-Wen Huang and his colleagues presented a tool named Web Application Vulnerability and Error Scanner (WAVES) [6] which used web-crawler to recognize all SQLIA holes, but it can not ensure integrity. In 2004, C. Gould, Z. Su, and P. Devanbu proposed the JDBC-Checker [7], however, its ability of solving SQLIA is limited. In the same year, S. W. Boyd put forward SQLrand [8], a technology based on digital signature, then decoding at runtime, if failure, it is subjected to SQLIA. But this way need to change the original business system deployment, it is also complex and unreliable. In 2005, Greg Buehrer and his colleagues presented SQLGuard [9] model that mainly deduced SQL statement tree at runtime, and then observed the result of the comparison with the structure of SQL statement tree. Although, this method needn't to identify SQL statement tree using the unique id, it generated the source tree many times. In 2010, Frankl et al. proposed a tool named ASSIST [10] for protecting Java-based web applications which could come from applications developed as JSPs or Servlets which presents the technique of automatic query sanitization to automatically remove SQL injection vulnerabilities in code. In 2011, Kadirvelu et al. proposed an intelligent dynamic query intent evaluation technique [11] to learn and predict the intent of the SQL queries provided by users and to compare the identified query structure with the query structure which has been generated with user input.

On the basis of the predecessors' research, we realize the combination of static and dynamic technology, and then put forward a new method of SQL Injection detection and defense based on AOP which can solve any SQLIAs effectively.

3 Proposed method

In this paper, we divided the SQLIAs into two kinds: One is this which have obvious attack characteristics, such as containing some sensitive and special attack characters. The other is the one that has no obvious characteristics. So, we go about to solve SQLIAs from two sides.

The first module, also named static defense module, designed to deal with some common SQLIAs which have obvious characteristics, such as acrossing the authorization, using select keyword and so on, by defining aspect and pointcut at which we obtained the external data using AOP. In other words, the application will execute the code that contained in pointcut which we have defined when a request arrives. And then doing some validations with the obtained parameters in the function of *before()*.

Figure 1 shows how the defined aspect will be applied into the general web project [12].

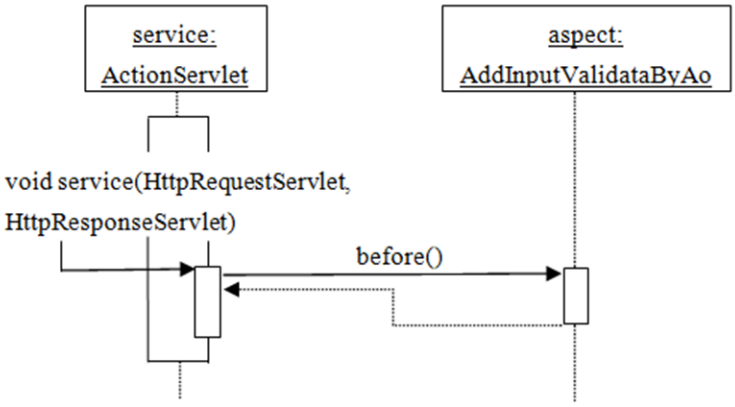

Figure 1. The application of AOP into web project.

The second module, also named real-time monitoring module. There is no rules to follow for unknown SQLIAs, although we can solve common attacks, that means it is far from enough to only rely on the static defense module. We learn that the essence of any types attack is to change the

structure of SQL which will be submitted to database, then we propose a method combined the static analysis, which aimed at analyzing and saving the original static SQL statement model before obtaining the external input data, with dynamic monitoring, which mainly realized the comparison with static SQL model and dynamic SQL String, and then weaving the monitoring code into source code by using AOP. Finally, we can judge how to do next according to the result of comparison.

Figure 2 shows the defense process combined the static analysis with dynamic monitoring.

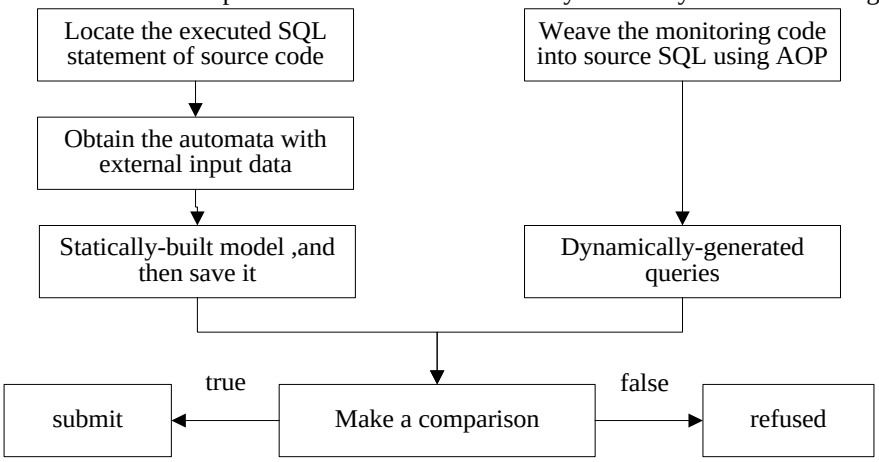

Figure 2. The process of combining static with dynamic.

4 Implementation

4.1 Static defense module

The main function of this module is doing some authentication operations aiming to the common SQLIAs which have obvious characteristics by using AOP. Figure 3 shows the process.

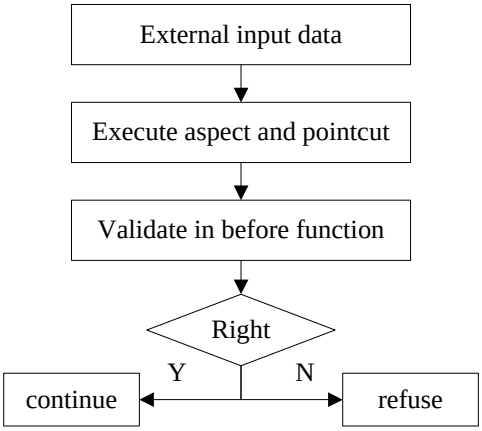

Figure 3. The validation process of static module.

Step1:There are generally including username and password when you enter the login page of website. Here you can attack website by constructing some malicious inputs;

Step2: Defining aspect in the background program, and then defining pointcut at the place where you obtained the data from external input. The format of pointcut is: *pointcut_excution()*;

Step3:Doing some validations for the obtained data in function of *before()*.

Then, we implement the static defense by using AOP technology, here is the core code:

```
public aspect AddInputValidateByAO {
    public pointcut captureHttpRequest(HttpServletRequest req,HttpServletResponse res) :
            execution(public void web.ActionServlet.service
                    (HttpServletRequest,HttpServletResponse)) && args(req,res);
   before(HttpServletRequest req,HttpServletResponse res)
              throws IOException: captureHttpRequest(req,res){
               String strName=request.getParameter("name"));
                    //doing some validations  }
}
```

4.2 Real-time monitoring module

We divided this module into two part. In its static part, program analysis technology is used to automatically build a model of legitimate queries. In its dynamic part, the monitoring code using AOP to check the dynamically-generated queries against the statically-built model is woven to application. Additionally, we can complete the interaction between static model and dynamic monitoring with the file of staticSQLModel.txt.

4.2.1 Static SQL model

Step1: First, we should analyze the interface which is interacted with background database by using the tool which can parse binary files, and then find out all connect Points. By analyzing interface of JDBC function, such as *java.sql. Statement. ExecuteQuery (String sql)*, we can learn the signature informations of function, then we may get the parameters position in SQL String, which named connectPoint. As shown below, line four is a connectPoint:

```
1    ……
2    String sql="select * from user_qw where ";
3    try {
4        sql+="username='"+userName+"' and pwd='"+password+"'";
5        rs=stmt.executeQuery(sql);
6            ……
7    }
```

Step2:According to the above information that we have got, we use the Java String Analysis (JSA) [13] , which constructs a flow graph that abstracts away the control flow of the program and represents string-manipulation operations performed on string variables. For each string of interest the technique analyzes the flow graph and simulates the string manipulation operations that are performed on the string. The result of the analysis is a Non-Deterministic Finite Automaton (NDFA) that expresses, at the character level, all the possible values the considered string can assume.

The character stream of connectPoint will be recorded after analyzing. As we all know, the structure of SQL may be different with different input that submitted by users, therefore, there are at least one automata state at each connectPoint.

Step3:For the automata mentioned above, we will read the state transition of each character, and traverse every possible full path of the characters flow, then we will get one or more String objects which will be saved into the file named staticSQLModel.txt. Meanwhile, it is essential to give an unique identification(connectPointId) for each connectPoints.

4.2.2 Real-time dynamic monitoring

The main function of real-time dynamic detection is monitoring the dynamic SQL String and weaving the monitoring code during execution process. Here is the procedure:

Step1:We choose the form of the static ordinary method matching section by using AOP. First, we should define a static method matching section, which matches all classes by default. Here, we should

define the pointcut in the form of the class filter and matching method name so that it can match the class what we want. Then, we need to declare a before enhancement class, also named before advice, in which we can add some logic of code which is needed.

When the program executes the function of *executeQuery()*, where will submit the SQL into database, we weave enhance code in before advice, meanwhile, we implement the comparison of dynamic and static SQL model string. Figure 4 shows the process.

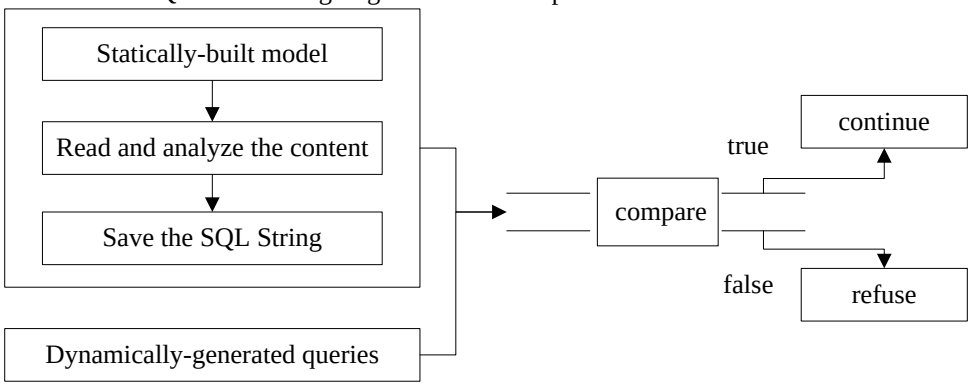

Figure 4. The process of comparison.

Step2:As shown above, during the comparison, *Detector* is the core part of real-time detection. In this part, there is a function named "*public Boolean Compare (int connectPointId, String dSql)*", which is mainly checking whether the dynamic SQL is matching with the collection of static SQL String which are related to the corresponding connectPointId. Addtionally, the first parameter, refers to the unique identification of connectPoint. The second, refers to the dynamic SQL which will be executed. The return value means the result of comparison.

First, we should accept the connectPointId through the function of *Compare()*, then find out the collection of static SQL Strings from the file of staticSQLModel. txt according to connectPointId. As mentioned above, there may be multiple SQL Strings with the same connectPointId, therefore, we should make a comparison with dsql and all corresponding static SQL strings. If the result matched only once, it is a normal request; otherwise, it is a SQLIA.

Finally, we can judge whether the request is valid or not according to the return value of the function of *Compare ()*. If result is true, continue. Else, refuse.

5 Experiment and analysis

In this paper, our method is implemented with java. As the SQL Injection Attacks always occur in the web application system, then we design a simple user login page for this test. First, we should insert the needed code of this method into background code, and then simulate the external input data and transmit into application, which will construct the dynamic SQL query string. Finally, we can know the defense capability of this method by the result of test.

In the login test page, the username and password in the SQL string are parameters obtained from external input, in which the attacker can construct some forms of attack statement, here we simulate three types of attacks including tautology, conjunctive query, compound query.

For the second defense module, at the beginning of the experiment, we should do some analysis for the static test object to get the automata model which is achieved by the tool of JSA. Figure 5 is a part of analysis results.

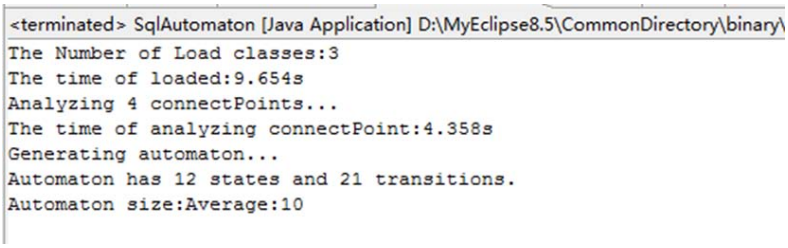

Figure 5. The result of static analysis.

We firstly analyze the corresponding class files of login page, then we can obtain the static SQL model in the source SQL code and save it into the file of staticSQLModel.txt by extracting the SQL keywords, identifiers and variables. During the experiment, the client will parse each command of test datas,then the request server received will experience the verification of two modules. Table 1 shows the defense ability of this method.

Table 1. The result of experiment.

Attack type	Attack characters	Result	Detected	Blocked
tautology	‘ or 1=1 -- ;	= is not valid	yes	yes
conjunctive query	‘ ;union select	Select is not valid	yes	yes
compound query	‘; drop table --	; is not valid	yes	yes
Illegal/incorrect	convert ()	Convert is not valid	yes	yes

As shown in the Table 1, there are four types of attacks presented, which included tautology, conjunctive, compound, and illegal query. In the table of experiment, it also listed the attack types, attack characters, and the result of detected and blocked, which can distinctly show that this method can detect and defense those different SQLIAs effectively.

6 Conclusion

In this paper, we proposed an AOP-based approach to the detection and defense of SQL injection attack on the basis of some analysis and summaries. As the result of this test, this method is able to prevent web application from all SQLIAs. However, there are some deficiencies, such as it needs to analyze the source code, that means the source program must be visible, which are my future work.

Acknowledgment

This research project was supported by National Natural Science Foundation of China under Grant No.61272115, 60873024.

References

1. OWASP Top10 Open Web Application Security Project. Top ten Web Application Secrity Risks. http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project,(2010)

2. Zhang Zhuo. SQL Injection Attack Technology and Countermeasures Analysis,(2007)
3. Menasce, D., Gomaa, H., Malek, S., and Sousa, J. (2011). SASSY: A Framework for Self-Architecting Service-Oriented Systems. IEEE Software **28**, 6, 78 –85.
4. C.A.Mackay. SQL Injection Attacks and Some Tips on How to Prevent Them. Technical report, The Code Project, http://www.codeproject.com/cs/database/Sqlias.asp, January (2005)
5. Hong Gui. Research on Aspect Conflict of Aspect-Oriented Software Development, (2007)
6. Chauhan, M. (2008). An Efficient Implementation of Candidate Evaluation in a Java Environment.https://alcazar.sisl.rites.uic.edu/wiki/pub/Main/CANDIDJavaImplementation/Project Report Megha.pdf.
7. C. Gould, Z. Su, and P. Devanbu. JDBC Checker: A Static Analysis Tool for Engineering (ICSE 04), pages 645–654, (2004).
8. S. W. Boyd, A. D. Keromytis. SQLrand: Preventing SQL Injection Attacks. In: Proceedings of the 2nd Applied Cryptography and Network Security (ACNS) Conference, June (2004). 292~302
9. G. T. Buehrer, B. W. Weide, and P. A. G. Sivilotti. Using Parse Tree Validation to Prevent SQL Injection Attacks. In International Workshop on Software Engineering and Middleware (SEM), (2005)
10. M.Raymond and F.Phyllis, "Preventing SQL Injection through Automatic Query Sanitization with ASSIST", Fourth International Workshop on Testing, Analysis and Verification of Web Software, EPTCS **35,** pp. 27–38, (2010)
11. K.Selvamani and A.Kannan, "ISQL-IDPS: Intelligent SQL-Injection Detection and Prevention System", European Journal of Scientific Research ISSN 1450-216X Vol.**51** No.2, pp.222-231,(2011)
12. Miles,R. AspectJ Cookbook in chinese, Tsinghua University Press,(2006)
13. Java String Analysis. http://www.brics.dk/JSA