# A Reusable SQL Injection Detection Method for Java Web Applications

**Chengwan He[1*] and Yue He[2]**
[1] School of Computer Science and Engineering, Wuhan Institute of Technology
Wuhan, Hubei 430205 - China
[e-mail: hechengwan@hotmail.com]
[2] School of Information Engineering, Wuhan University of Technology
Wuhan, Hubei 430000 - China
[e-mail: 1096853550@qq.com]
*Corresponding author: Chengwan He

*Received July 29, 2019; revised March 2, 2020; accepted April 26, 2020; published June 30, 2020*

## Abstract

The fundamental reason why most SQL injection detection methods are difficult to use in practice is the low reusability of the implementation code. This paper presents a reusable SQL injection detection method for Java Web applications based on AOP (Aspect-Oriented Programming) and dynamic taint analysis, which encapsulates the dynamic taint analysis processes into different aspects and establishes aspect library to realize the large-grained reuse of the code for detecting SQL injection attacks. A metamodel of aspect library is proposed, and a management tool for the aspect library is implemented. Experiments show that this method can effectively detect 7 known types of SQL injection attack such as tautologies, logically incorrect queries, union query, piggy-backed queries, stored procedures, inference query, alternate encodings and so on, and support the large-grained reuse of the code for detecting SQL injection attacks.

***Keywords:*** SQL injection attack, aspect-oriented programming, taint analysis, aspect library, metamodel

## 1. Introduction

The increasing security of Web applications has been a challenging topic for scholars and software developers. There are mainly the following reasons [1]: 1) Technical Challenges. Software developers are not all security experts, and even the best-trained developers can focus on only a few security issues at a time. 2) Aggressive Adversaries. Hackers have realized that attacking application vulnerabilities are often the fastest and easiest way. Statistics show that 37% of security problems occur in the application layer, and over 65% of organizations have suffered SQL injection attacks [2]. 3) Organizational Factors. Developers and quality assurance (QA) engineers are usually rewarded for their ability to deliver features quickly, not for discovering and eliminating security flaws.

The above reasons indicate the following facts: 1) For most developers that are inexperienced and have no knowledge of security programming techniques, even if the security experts point out the program defects, there is no way to solve these security defects. Programmers need to reuse solutions, technologies, and especially code to prevent security threats at large granularity. 2) Application vulnerabilities are the main targets of hacker attacks, Web applications should use more general and robust detection and defense methods, and tackle SQL injection attacks as the main target of detection and defense; 3) In order to shorten the development cycle and reduce the development cost, most enterprises do not fully analyze and mitigate the security vulnerabilities that may be contained in the program, which leads to the result that a large number of Web applications may suffer from security vulnerabilities. Reworking all the code is almost impossible, and dynamic attack detection and defense are essential to complement static security detection.

SQL injection attacks come in many forms. In 2006, according to the purpose and intention of attackers, Halfond et al. divided SQL injection attacks into 7 types, including tautologies, logically incorrect queries, union query, piggy-backed queries, stored procedures, inference query, alternate encodings and so on [3].

To prevent SQL injection attacks, many scholars have published a lot of fruitful research results, including static detection methods in the phase of implementing code, and dynamic detection methods during the stage of code deployment and execution. However, most methods have certain limitations. The shortcomings of existing methods mainly include:

Most existing methods can only detect a part of the known types of SQL injection attacks, the accuracy of SQL injection detection needs to be improved.

The detection code cannot be reused with large granularity. There are various types of SQL injection attacks, and their detection and defense require specific expertise. Most programmers may not have such knowledge and skills. Although programmers can mitigate security vulnerabilities in code to some extent after professional training in security, how to reduce the cost for developing secure code in a short software development cycle is an issue that enterprise managers must consider. Most of the existing methods lack the mechanism of large-grained reuse of the code for SQL injection detection.

In this paper, an approach based on dynamic taint analysis [4]-[8] and AOP (Aspect-Oriented Programming) [9] is proposed to detect and defend the external attacks of software at runtime, and realize the large-grained reuse of SQL injection detection code. We use AOP to encapsulate dynamic taint analysis processes (taint marking, taint propagation analysis, SQL syntax analysis), and build an aspect library for SQL injection detection to achieve the reuse of large-grained aspect code. Different from normal class libraries or

function libraries, the interrelated aspects used for tainting and taint propagation analysis can be reused as a whole.

The main contributions of this paper are the following.

(1)A dynamic taint analysis method based on AOP is presented, including taint marking, taint propagation and taint analysis. In order to improve the detection accuracy, we propose a new taint marking method, which not only marks the user's input as the tainted data, but also marks the starting and ending positions of the tainted data in a string, which realizes accurate analysis of the tainted data.

(2)This paper proposes a metamodel of aspect library. Through the construction of an aspect library and its management tool, the large-grained reuse of SQL injection detection code is realized.

The approach proposed in this paper is an improvement and further study of our previous work [10]. In this paper, we improve the tainting method of starting and ending information of tainted data, and propose the taint propagation algorithm using string concatenation and substring extraction as examples. At the same time, we also propose the metamodel and construction method of aspect library.

The rest of the paper is structured as follows: In Section 2, we introduce the related work of SQL injection detection. Then, we describe an SQL injection detection method based on AOP and dynamic taint analysis in Section 3. An metamodel of Aspect library is presented in Section 4. In Section 5, we use some experiments to evaluate the proposed method and in Section 6, we conclude this paper.

## 2. Related Work

### 2.1 Vulnerability Detection Based on Program Analysis

Static analysis technology analyzes the source code of Web application lexicographically, syntactically, and data processing process and logical structure, to find possible security risks in the coding phase of the software development life cycle. Because of the context-free nature of SQL statements, the syntax and semantics of normal and malicious statements are quite different, so the syntax analysis technique can be applied to the detection of SQL injection behavior. For static analysis, it is usually necessary to use a vulnerability scanner to conduct a white-box scan of source code to determine the type of software vulnerability and corresponding code block. For example, Shinetal's SQLUnitGen [11] located risk points and generated inspection report through an automatic test, and developers manually modified the defect code according to the scan results to repair software vulnerabilities. But for most developers that are inexperienced and have no knowledge of security programming techniques, there is little they can do about the results. Dynamic analysis method [12] includes data flow analysis and control flow analysis, which find the vulnerabilities by observing the dynamic properties such as memory usage and register values during program execution. The data flow analysis attempts to trigger its potential vulnerabilities by constructing the boundary data, and the control flow analysis detects the defects of the function call by setting the breakpoint in real-time to track the control state transformation of the target program.

### 2.2 Vulnerability Detection Based on Taint Analysis

Researchers try to reason the trusted and untrusted parts of SQL statements according to a certain strategy, but the accuracy of detection cannot be guaranteed. Taint analysis is slightly more reliable than the above methods in accurately distinguishing trusted from untrusted parts

through tainting, taint propagation analysis and harmless processing, which is mainly divided into dynamic and static taint analysis. In static taint analysis, Lam et al. used static data flow analysis technology to check whether the tainted data inputted by the user reached the execution point of SQL statement to dig SQL injection vulnerabilities [13]. Pixy [14] proposed by Jovanovic et al. uses flow-sensitive, inter-process and context-sensitive static data flow analysis methods for PHP programs to detect whether user input in target statements has been properly verified and processed, thus mining Web application vulnerabilities. Minamide proposed a string analysis method for PHP programs to accurately and efficiently simulate dynamic Web content [15]. Wassermann et al. [16], based on the string analysis method proposed by Minamide, used context-free grammar (CFG) to abstract the construction. process of SQL statement, determined whether the grammar non-terminator was related to user input with the static taint analysis method, and detected SQL injection vulnerability by tracking it. Su also modified the string analysis method to make it suitable for static detection of XSS vulnerabilities [17]. However, because this kind of static analysis needs to rely on traditional program analysis methods and cannot well deal with the dynamic characteristics of Web applications, it produces high false positives and false negatives and increases the burden of manually confirming vulnerability results, so it can handle very limited types of programs.

Dynamic taint analysis [18]-[22] can better deal with the dynamic characteristics of the program and can be divided into negative tainting and positive tainting based on untrusted data and trusted data. For example, Nguyen et al. [23] proposed to dynamically track the propagation process of taint mark at the character granularity level by modifying the data structure of String class and built-in operation function in the PHP interpreter, and analyze the syntax structure of SQL statement at the execution point (sink), not allowing SQL keywords and sensitive characters to come from untrusted data. Qed [24] is a goal-oriented model detection system that can automatically generate test cases to test Java programs with possible vulnerabilities. CSSE [25] creates index tables with pointer variables as indexes in the PHP engine and implements character level tainting for untrusted data with the bitmap as a value type. Ardilla [26] is able to automatically generate test cases for XSS and SQL injection in PHP, traces taints through symbol execution and generates specific vulnerability exploit code based on changes in taint information. All the above approaches choose to mark the user input as the tainted data and propagate the taint mark when the program runs. In practice, not only are the sources and types of untrusted data widely distributed, but it is also difficult to accurately track each of the tainted variables. In contrast, trusted data sources are more fixed, such as hard-coded strings that programmers write into programs. Based on this characteristic, researchers proposed a positive tainting method. Halfond et al. [4] set the trusted mark for hard-coded strings in the program and dynamically tracked the propagation track of the mark. The trusted and untrusted parts in SQL statements are distinguished by this mark. Since all untrusted inputs exist in unmarked form, this method can effectively avoid underreporting the attacks and improve the accuracy of SQL injection attack detection. All of the above methods record taint information by allocating extra memory space or modifying data structure, Although dynamic taint propagation analysis can give accurate results, it relies on effective rules and string constraint solver accuracy to reduce the false negative, In other words, this approach increases program coverage and increases the performance penalty of related functions.

### 2.3 Other Approaches

In recent years, researchers have proposed character/encoding conversion, boundary marking and character randomization to represent trusted/untrusted data to implement lightweight taint

analysis. PHPHard [7] is a method based on a similar idea of boundary tags. It uses positive tainting to mark the inline HTML code and constant strings in PHP, and dynamically tracks the security interval of response pages through the propagation of boundary tags, so as to detect cross-site scripting attacks. SQLrand [27] is a protection scheme of the randomizing instruction set. It adds a random integer after the keywords in the hard-coded string to carry out the randomizing process to mark the trusted keywords, checks the legitimacy of the SQL statement through the agent, and passes it to the database for execution after de-randomizing. Zhang Huilin et al. proposed a tainting method based on encoding conversion [28], which is applicable to UTF-8 encoded Web applications. The taint information is stored directly in the non-standard single-byte encoding, the string length after the tainting is not affected, and the tainted mark automatically carried in the character byte will be automatically propagated with the program execution, and no additional taint tracking algorithm is required. Zhao Yufei et al. took network traffic as the training data [29] to extract the features of malicious requests from the real network environment to detect SQL injection.

## 3. Method Design and Implementation

The basic idea of this method is shown in **Fig. 1**. Each process of the taint analysis is encapsulated in different aspects and stored in the aspect library. Different applications can instantiate abstract aspects in the aspect library to achieve large-grained reuse of taint analysis.

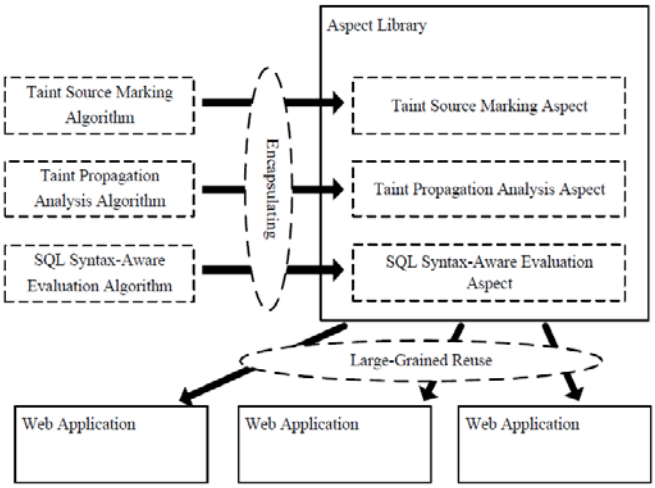


**Fig. 1.** The basic idea

The SQL injection detection method based on AOP and dynamic taint analysis is shown in **Fig. 2**. Taint source marking aspect marks the input data of the system as tainted data. Taint propagation analysis aspects track string manipulation functions and annotate operation results. The syntax-aware evaluation aspect evaluates the SQL syntax and detects the presence of an SQL injection attack string before submitting the SQL statement to the database for execution.

### 3.1 Taint Source Marking Aspect

Tainted data (untrusted data) includes many types. A typical example of tainted data is the data inputted by the user in the interface, by which an attacker inputs a string containing a special form and submits it to the Web server for execution to steal the information. Also, cookies

received by the server and data read from the file may contain attack strings.

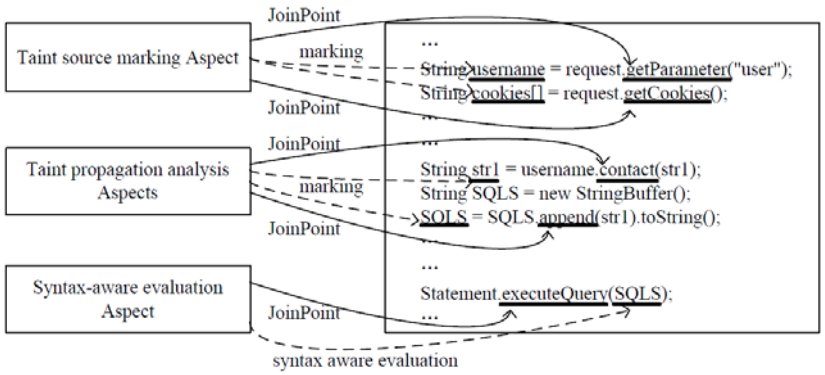


**Fig. 2.** The SQL injection detection method based on AOP and dynamic taint analysis

To mark the tainted data, we add a field taintIndex to String, StringBuffer and StringBuilder classes in JDK to mark the tainted information. After string manipulation, if a string contains more than one tainted substring, taintIndex will contain the starting and ending positions of these tainted substrings.

```
public final class String {
    private final char[] value;
    …
    private final Hashtable taintIndex;
}
```

The tainting process is shown in **Fig. 3**. The taint marking aspect takes the user's input data as the taint marking source and sets the taintIndex to [0, userinputdata.leng-1].

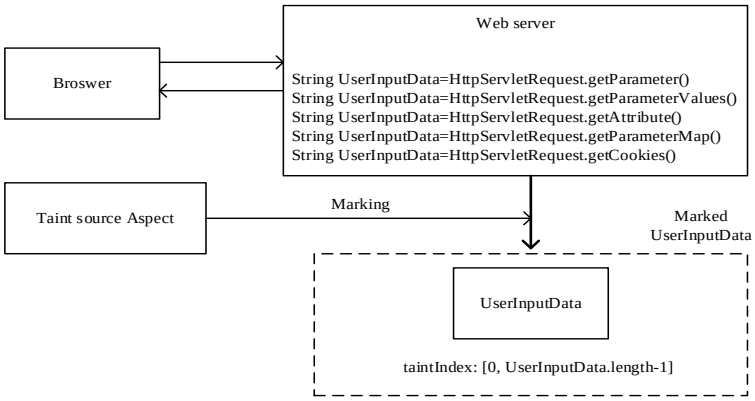


**Fig. 3.** Tainting process

### 3.2 Taint Propagation Analysis Aspects

String manipulation functions (such as string concatenation and splitting functions) and assignment statements have the property of taint propagation. AOP-based taint propagation analysis defines string manipulation functions and assignment operation as join points. In each join point, the analysis process of taint propagation is woven separately(**Fig. 4**).

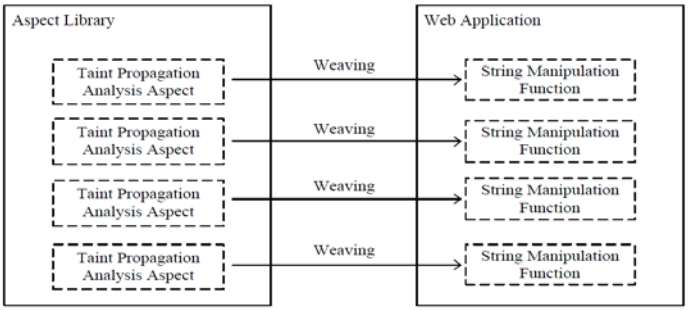


**Fig. 4.** Taint propagation analysis

According to the Java function manual, there are 216 basic string functions. Through careful analysis, 81 of them have the property of taint propagation. For these functions, the logic of taint propagation should be implemented one by one. This paper selects two commonly used string manipulation functions (string concatenation and substring extraction) to illustrate the algorithm of taint propagation analysis.

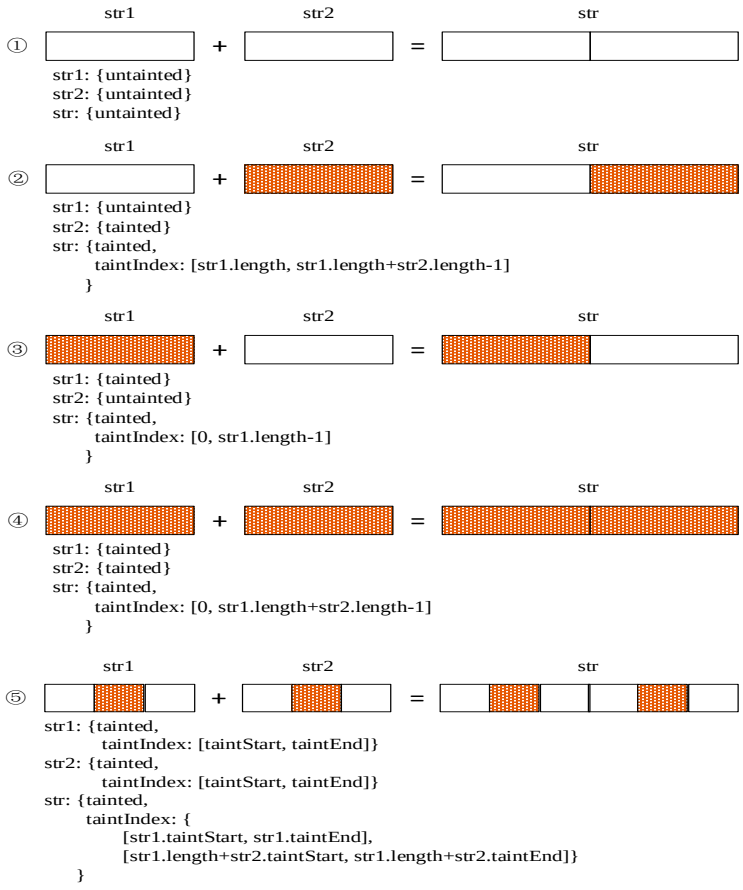


**Fig. 5.** String concatenation

**Fig. 5** shows the taint propagation analysis of the string concatenation operation. Depending on the string being manipulated, the analysis process can be divided into five scenarios:

(1) If neither of the manipulated strings is a tainted string, then the result string does not

contain tainted data.

(2) If the first manipulated string is not a tainted string and the second manipulated string is a tainted string, then the result string contains tainted data, and the index of starting and ending positions of the tainted data is [str1.length, str1.length+ str2.leng-1].

(3) If the first manipulated string is a tainted string, and the second manipulated string does not contain tainted data, then the result string contains tainted data, and the index of starting and ending positions of the tainted data is [0, str1.leng-1].

(4) If both of the manipulated strings are tainted strings, then the result string contains tainted data, and the index of starting and ending positions of the tainted data is [0, str1.length+ str2.leng-1].

(5) If both manipulated strings contain partial tainted strings, then the result string contains tainted data, and the index of starting and ending positions of the tainted data is [str1.taintStart, str1.taintEnd] and [str1.length+str2.taintStart, str1.length+str2.taintEnd].

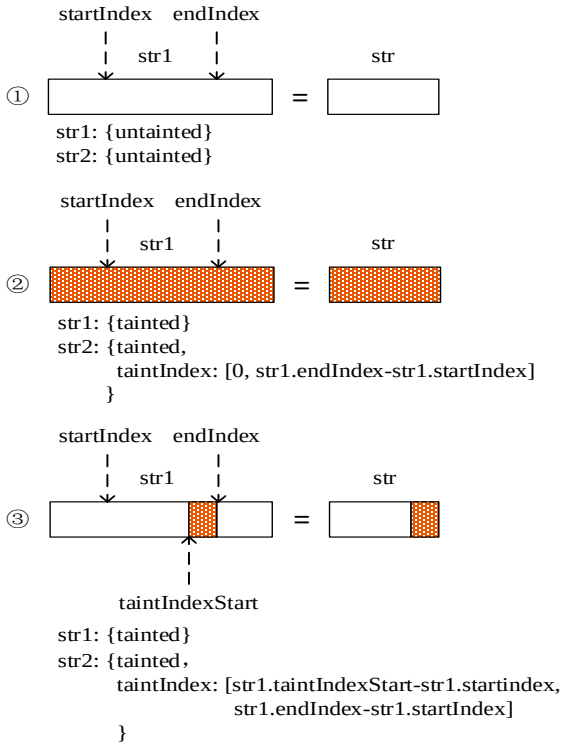


**Fig. 6.** Substring extraction

**Fig. 6** shows the analysis process of taint propagation of the substring extraction operation. Depending on the string being manipulated, the analysis process can be divided into three scenarios:

(1) If the manipulated string is not a tainted string, then the substring is not a tainted string.

(2) If the manipulated string is a tainted string, then the substring is a tainted string, and the index of starting and ending positions of the tainted data is [0, str1.endindex-str1.startindex].

(3) If the manipulated string contains a partial tainted string, then the substring is a tainted string, and the index of starting and ending positions of the tainted data is [str1.taintIndexStart-str1.startindex, str1.endIndex-str1.startIndex].

### 3.3 Taint Detection Strategy

All related operations in Java programs that interact with the database are encapsulated in the JDBC library. SQL injection sinks refer to the methods for accessing database, such as:

Statement.executeQuery()
Statement.executeUpdate()
Connection.prepareStatement()

The core of the detection method is to check whether the structure of the SQL statement constructed by the application after receiving data inputted by user has changed. For accurate detection, we use ANTLR language recognition tool to parse dynamically captured SQL statements.

The SQL syntax evaluation resolves the SQL string recursively into three tokens corresponding to keywords, operators, and literals.

**Table 1.** SQL keywords

| Type | Samples |
|---|---|
| Function | SLEEP,SUBSTRING |
| Keyword | FROM,WHERE |
| Expression | UPDATE,SELECT |
| Union | UNION |
| Type | INT |
| Operator | > |
| Comment | #,-- |
| Variable | Var |

If it is found that the parsed SQL statement does not contain any non-trusted syntax related characters, it indicates that the statement conforms to the SQL security conditions, there is no SQL injection behavior, and it is sent directly to the database for execution. If any of the lexical symbols shown in **Table 1** are tainted after parsing, these untrusted syntax-related characters can break the logical structure of the original SQL statement, causing SQL injection attacks. Automatically defend against SQL injection attacks by adding an escape character "\" before these non-trusted syntax-related characters to signal an alternative interpretation of the original characters.

## 4. Aspect Library Construction

### 4.1 Definition of Abstract Aspects

Because third-party libraries may be used in Web applications, string manipulation functions with the same functionality may have different names for different libraries. To make processes such as taint source marking and taint propagation analysis independent of libraries or frameworks used by Web applications, we define join points based on the semantic information of string manipulation functions, rather than using specific function names. When instantiating an abstract aspect, the specific join point definitions are automatically generated using the defined semantic mapping table (**Fig. 7**).

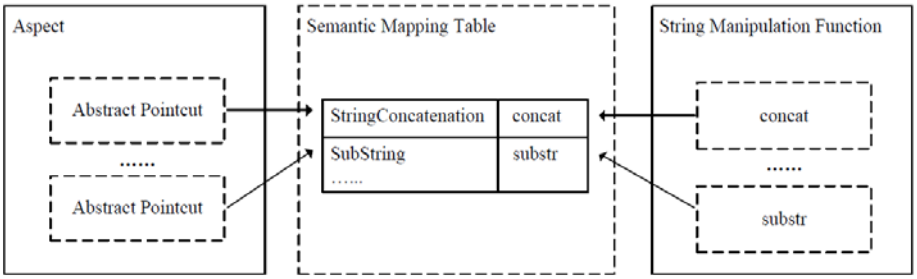


**Fig. 7.** Join points definition based on the semantic information

### 4.2 Metamodel of Aspect Library

From the aspects of file organization, information description, and reusable aspect definition method, we propose a general reusable construction method of aspect library, design and implement the management tool of aspect library, which is used to manage, retrieve and automatically generate concrete aspects of reusable abstract aspects, and improve the efficiency of aspect reuse. **Fig. 8** is our proposed metamodel of aspect library , which mainly includes three parts: source code of abstract aspects, description XML file of aspect library, and management tool of aspect library.

(1)Source Code of Abstract Aspect

Source code of abstract aspect is mainly organized by different functional categories. There are three types of aspects: taint source marking, taint propagation analysis, and SQL syntax analysis. In addition to AspecJ's basic elements that define abstract aspects: aspect name, abstract pointcuts, advices, inter-type declarations, methods, attributes, and so on, the internal structure of each abstract aspect also includes aspect-description annotations, abstract pointcut description annotations, and so on.

(2)Aspect Library Description XML File

An aspect library description XML file is an XML file defined for describing an aspect library and used to document the basic information of the aspect library, reusable aspect information, and reusable definitions of pointcuts. Aspect library description XML files are automatically managed by the management tool, including saving aspect information when loading source code of abstract aspects and saving concrete definitions of pointcuts when instantiating abstract pointcuts.

(3)Management Tool of Aspect Library

Management tool of aspect library provides an ancillary tool for developers to efficiently manage reusable aspect library resources. Its main functions are to load, manage and retrieve source code of aspect library and automatically generate concrete aspects.

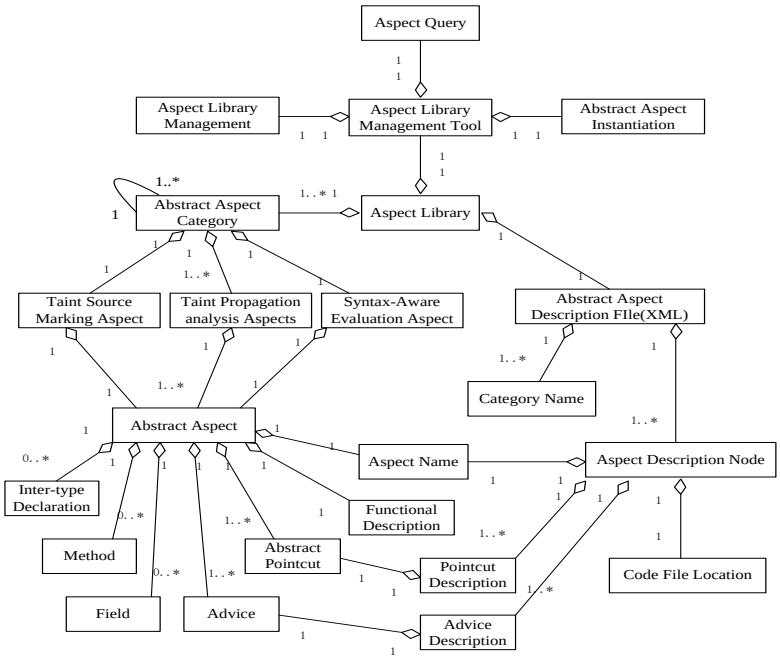


**Fig. 8.** Metamodel of Aspect library

## 5. Experiment and Discussion

Through manual and automated tools, injection behavior requests are frequently sent under each injection point of the WAVSEP(Web Application Vulnerability Scanner Evaluation Project) [30] and WebGoat [31] target environment to simulate the attacker's behavior. The detection results are statistically analyzed in **Table 2**. Compared to several other typical methods, our method can detect all seven known types of SQL injection attacks. When performing the taint analysis at the sink, we can judge whether the SQL statement contains a SQL injection attack based on the taint marking information and the result of the SQL syntax analysis, which improves the accuracy of detection.This approach should also be effective for unknown types of SQL injection attacks. At the same time, we can accurately replace illegal characters in the SQL statement based on the starting and ending position information of the tainted data, so as to achieve the purpose of automatic defense.

**Table 2.** Comparison of Detection Types of Several Typical Methods

| Approaches | Tautologies | Logically Incorrect Queries | Union Query | Piggy-Backed Queries | Stored Procedures | Inference Query | Alternate Encodings |
|---|---|---|---|---|---|---|---|
| CSSE[24] | √ | √ | √ | √ | × | √ | × |
| SQLCHECK[16] | √ | √ | √ | √ | × | √ | √ |
| Our Approach | √ | √ | √ | √ | √ | √ | √ |

The portability of the SQL injection detection model proposed in this paper is reflected in the following three levels (shown in **Fig. 9**):

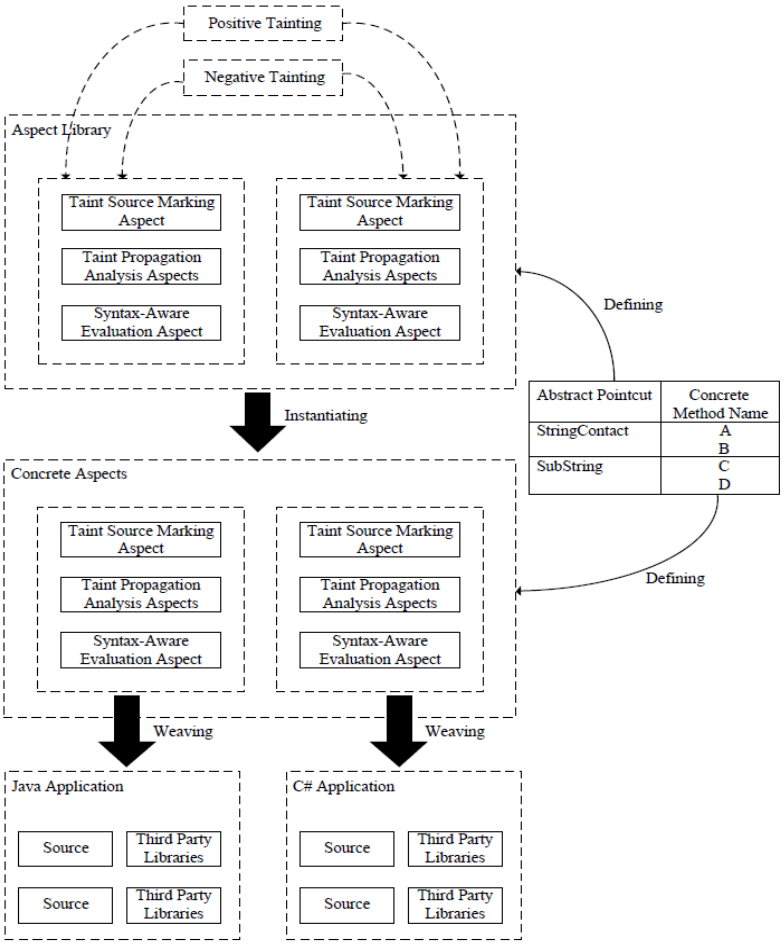


**Fig. 9.** Method portability

(1) Portability For Different Tainting Methods

Dynamic taint analysis has two methods: positive tainting and negative tainting, both of which have advantages and disadvantages. Considering that the type of taint source on the server-side of a Web application is limited, and our method allows users to specify the taint source, this paper adopts the method of negative tainting and uses AOP to encapsulate tainted data and taint propagation analysis. However, we believe that this method is also suitable for the taint analysis process with positive tainting, except that the implementation code of the tainted data marking and SQL syntax analysis are slightly different.

(2) Portability for Third-party Libraries

Today's software development is inseparable from the third-party libraries (including open source and free third-party libraries), even if the functions that implement the same function, different libraries and software frameworks may provide different APIs. For example, string concatenation function, in some libraries the name of the function is A, while in other libraries the name is B. In order to make the taint propagation analysis correctly, we use the semantic information (an abstract concept) of the string manipulation function to define join points in the abstract aspects, and then use the corresponding specific function name to define the join

points when generating the concrete aspects, so as to ensure the correctness of the analysis of taint propagation.

(3) Portability for Different Programming Languages

Aspects implemented in one language cannot be directly applied to Web applications developed by other programming languages, but our detection model is completely portable as long as that language has the corresponding AOP implementation or the Weaving mechanism for the aspects.

## 6. Conclusion

This paper presented a reusable SQL injection behavior detection method for Java Web applications based on AOP and dynamic taint analysis, which encapsulates the dynamic taint analysis processes into different aspects and establishes the aspect library to realize the large-grained reuse of SQL injection detection code. The detection code encapsulated by aspect is woven into the source code by the weaver. The process does not need to modify the execution engine of application and source code and has strong portability for applications using different programming languages. This method is also applicable to other code injection attacks such as cross-site scripting (XSS) and cross-site request forgery (CSRF).

## References

[1] IBM Security. “Five Steps to Achieve Risk-Based Application Security Management,” *Thought Leadership White Paper*, Jul. 2015.

[2] L. K. Shar, H. B. K. Tan, “Defeating SQL injection,” *Computer*, vol. 46, no. 3, pp. 69-77, 2013. Article (CrossRef Link).

[3] W. G. J. Halfond, J. Viegas, and A. Orso, “A classification of SQL injection attacks and countermeasures,” in *Proc. of the International Symposium on Secure Software Engineering, Washington, USA*, pp. 13-15, 2006.

[4] W. G. J. Halfond, A. Orso, P. Manolios, “WASP: Protecting Web Applications Using Positive Tainting and Syntax-Aware Evaluation,” *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, vol.34, no.1, PP. 65-81, 2008. Article (CrossRef Link).

[5] M. Sridharan, S. Artzi, M. Pistoia, S. Guarnieri, O. Tripp, and R. Berg, “F4F: Taint analysis of framework-based Web applications,” *ACM SIGPLAN Notices*, vol. 46, no. 10, pp. 1053−1068, 2011. Article (CrossRef Link).

[6] I. Papagiannis, M. Migliavacca, and P. Pietzuch, “PHP ASPIS: Using partial taint tracking to protect against injection attacks,” in *Proc. of the Usenix Conf. on Web Application Development*, pp. 1-8, Feb. 2011.

[7] WANG Yi, LI Zhou-jun, and GUO Tao, “Literal tainting method for preventing code injection attack in web application,” *Journal of Computer Research and Development*, vol. 49, no.11, pp. 2414-2423, 2012.

[8] WANG Lei, LI Feng, LI Lian, et al, “Principle and practice of taint analysis,” *Journal of Software*, vol. 28, no. 4, pp. 860-882, 2017. Article (CrossRef Link).

[9] G. Kiczales, J. Lamping, A. Mendhekar, et al, “Aspect-oriented programming,” in *Proc. of the European Conference on Object-Oriented Programming, Jyvaskyla, Finland*, pp. 220-242, 1997. Article (CrossRef Link).

[10] HE Cheng-wan, YE Zhi-peng, “SQL Injection Behavior Detection Method Based on AOP and Dynamic Taint Analysis.” *Acta Electronica Sinica*, vol.47, no.11, pp.2413-2419, 2019. Article (CrossRef Link).

[11] Y. Shin, L. Williams, T. Xie, "SQLUnitGen: Test Case Generation for SQL Injection Detection," *North Carolina State University*, 2006.
[12] M. S. Lam, M. Martin, J. Whaley, et al, "Securing web applications with static and dynamic information flow tracking," in *Proc. of ACM Sigplan Symposium on Partial Evaluation and Semantics-Based Program Manipulation, San Francisco, CA, USA*, pp.3-12, 2008. Article (CrossRef Link).
[13] V. B. Livshits, and M. S. Lam, "Finding security vulnerabilities in java applications with static analysis," in *Proc. of the 14th Conference on USENIX Security Symposium, California, USA*, pp. 18-18, 2005.
[14] N. Jovanovic, C. Kruegel, and E. Kirda E, "Pixy: a static analysis tool for detecting web application vulnerabilities," in *Proc. of IEEE Symposium on Security and Privacy*, pp. 258-263, Berkeley, USA, 2006. Article (CrossRef Link).
[15] Y. Minamide, "Static approximation of dynamically generated Web pages," in *Proc. of the International Conference on the World Wide Web*, pp. 432-441, 2005.
[16] G. Wassermann, and Zhendong Su, "Sound and precise analysis of web applications for injection vulnerabilities," in *Proc. of PLDI '07: Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 32-41, 2007. Article (CrossRef Link).
[17] G. Wassermann, Zhendong Su, "Static detection of cross-site scripting vulnerabilities," in *Proc. of ACM/IEEE International Conference on Software Engineering*, pp. 171-180, 2008. Article (CrossRef Link).
[18] A. Naderi-Afooshteh, A. Nguyen-Tuong, M. Bagheri-Marzijarani, et al, "Joza: Hybrid taint inference for defeating web application SQL injection attacks," in *Proc. of IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 172-183, Rio de Janeiro, Brazil. Article (CrossRef Link).
[19] E. J. Schwartz, T. Avgerinos, and D. Brumley, "All You Ever Wanted to Know about Dynamic Taint Analysis and Forward Symbolic Execution (but Might Have Been Afraid to Ask)," in *Proc. of IEEE international Conference on Security and Privacy*, pp. 317-331, 2010. Article (CrossRef Link).
[20] ZHOU Ying, FANG Yong, HUANG Cheng, et al, "Detection of SQL injection behaviors for PHP applications," *Journal of Computer Applications*, vol. 38, no. 1, pp. 201-206, 2018.
[21] P. Vogt, F. Nentwich, N. Jovanovic, E. Kirda, C. Kruegel, G. Vigna, "Cross site scripting prevention with dynamic data tainting and static analysis," in *Proc. of the Network and Distributed System Security Symposium, San Diego, California, USA*, Feb. 2007.
[22] MA Jin-xin, LI Zhou-jun, ZHANG Tao, et al, "Taint analysis method based on offline indices of instruction trace," *Journal of Software*, vol. 28, no. 9, pp. 2388-2401, 2017. Article (CrossRef Link).
[23] A. guyen-Tuong, S. Guarnieri, D. Greene, et al, "Automatically hardening web applications using precise tainting," in *Proc. of IFIP 20th International Information Security Conference, Chiba, Japan*, pp. 295-307, 2005. Article (CrossRef Link).
[24] M. Martin, M. S. Lam, "Automatic Generation of XSS and SQL Injection Attacks with Goal-directed Model Checking," in *Proc. of USENIX Security Symposium*, pp. 31-44, 2008.
[25] T. Pietraszek, C. V. Berghe, "Defending against injection attacks through context-sensitive string evaluation," in *Proc. of International Conference on Recent Advances in Intrusion Detection, Seattle, WA, USA*, pp. 124-145, 2005. Article (CrossRef Link).
[26] A. Kieyzun, P. J. Guo, K. Jayaraman, et al, "Automatic creation of SQL Injection and cross-site scripting attacks," in *Proc. of IEEE International Conference on Software Engineering, Vancouver, BC, Canada* , pp. 199-209, 2009. Article (CrossRef Link).
[27] S. W. Boyd, and A. D. Keromytis, "SQLrand: Preventing SQL Injection Attacks," in *Proc. of 2ndInternational Conference on Applied Cryptography and Network Security, Yellow Mountain, China*, pp. 292-302, 2004. Article (CrossRef Link).
[28] ZHANG Hui-lin, DING Yu, ZHANG Li-hua, et al, "SQL injection prevention based on sensitive characters," *Journal of Computer Research and Development*, vol. 53, no. 10, pp. 2262-2276, 2016. Article (CrossRef Link).

[29] ZHAO Yu-fei, XIONG Gang, HE Long-tao, et al, "Approach to detecting SQL injection behaviors in network environment," *Journal on Communications*, vol. 37, no. 2, pp. 89-98, 2016. Article (CrossRef Link).
[30] Shay Chen, "The Web Application Vulnerability Scanner Evaluation Project," 2019.
[31] OWASP, "WebGoat," 2019. [Online]. Available: https://github.com/WebGoat/WebGoat,

**Chengwan** received master's degree in Information Engineering of Hokkaido University in Japan. In 2005, he received doctor's degree in Computer Software and Theory of State key laboratory of Software Engineering, Wuhan University, China. His current research interest is theory and application of software engineering.

**Yue He** is a junior student in Wuhan University of Technology and majors in Electricity Engineering currently. His current research interest is digital signal processing algorithms and applications.