Journal of Information Hiding and Multimedia Signal Processing ©2017 ISSN 2073-4212
Ubiquitous International Volume **8**, Number **4**, July **2017**

A Parse Tree-Based NoSQL Injection Attacks Detection Mechanism

Hong Ma, Tsu-Yang Wu*, Min Chen, Rong-Hua Yang, and Jeng-Shyang Pan

Fujian Provincial Key Laboratory of Big Data Mining and Applications
National Demonstration Center for Experimental Electronic Information and Electrical Technology Education
College of Information Science and Engineering
Fujian University of Technology
No3 Xueyuan Road, University Town, Fuzhou 350118, China
wutsuyang@gmail.com; chenmin@fjut.edu.cn; 287456@qq.com; jengshyangpan@fjut.edu.cn
*Corresponding author's email: wutsuyang@gmail.com

Received June, 2017; revised June, 2017

Abstract. *Nowadays, many IT giants such as Facebook, Google, and Amazon adopt non-relational database (NoSQL, Not only SQL) technologies to manage their systems. Although these kind of database technologies have made outstanding contributions to the development of the IT industry, it also exposed some security risks such as SQL injection attacks. Up to now, there are many solutions to counter SQL injection attacks in SQL databases. However, there exist few approaches to counter injection attacks in NoSQL databases. So, how to design an effective NoSQL injection attacks detecting mechanism becomes a subject worthy of in-depth study. In this paper, we based on semantic structure analysis of execution statements to propose a detection approach using parse tree. Based on this approach, we focus on MongoDB to propose a dynamic NoSQL injection attacks detection mechanism in the web environment called DND. It does not require access to or modifying source codes, rewriting source codes with extra libraries, or complex assisted devices. Finally, the experimental results are shown that DND has high accuracy rates, low false positive rates, and low response time.*

Keywords: Nosql injection attacks, Parse tree, Detection, Web environment, MongoDB

1. **Introduction.** In the past twenty years, traditional relational databases (SQL) such as MySQL, Oracle, DB2, and SQL server have provided better services to IT companies. With fast growing of the mobile Internet traffic, SQL databases are not the best choices for IT companies [1, 2]. Thus, NoSQL databases such as MongoDB, Redis, and Cassandra with a low cost and scalability quickly becomes a new issue in IT industries [3, 4]. Many IT giants such as Facebook, Google, and Amazon adopt NoSQL technologies to manage their database systems. Although these kind of database technologies have made outstanding contributions to the development of the IT industries, it also exposed some security risks.

SQL injection attack was first introduced by Rain Forest Puppy [5] in 1998 and becomes one of the security threats in backend applications [6, 7, 8, 9]. It allows attacker can launch SQL injection attack to execute a lot of malicious operations on the database such as bypassing authentication, access privacy information, changing, deleting, and adding new data. Although the maturity of the security framework and the gradual increase of security awareness such that the occurring of SQL injection attacks has been decreasing recently, it is still the most common attack method. As we all know, SQL injection attacks detection techniques had been developed completely. There are several methods such as static analysis [10, 11, 12], dynamic analysis [13], static and dynamic analysis [14, 15], machine learning [16, 17, 18], taint tracking [19, 20, 21, 22], and parse tree based [23, 24, 25].

Now, NoSQL databases suffered from same risks in SQL databases. One possible way is to modify the SQL injection attacks detection methods and then deploys them to NoSQL databases. However, to deploy SQL injection attacks detection methods to NoSQL databases is unrealistic for example NoSQL databases do not support SQL language. It removes SQL injection attacks in NoSQL databases naturally. Meanwhile, NoSQL databases are very flexible. For example, MongoDB uses MONGO Shell to execute add, delete, modifying, and find operations and supports JavaScript. For different programming languages, NoSQL databases provide different interfaces. It shows the difference between SQL and NoSQL databases.

In other aspect, these NoSQL databases adopt different query statements so that the traditional SQL injection attacks have no effect on them. It means that NoSQL databases are secure against SQL injection attacks? The answer is No [26]. Like most new technologies just emerging, NoSQL database is missing some security mechanisms such as encryption, authentication, and role management[27]. As a result, NoSQL databases are also suffered from DOS or DDOS attacks, injection attacks, and so on. Therefore, it is necessary to design NoSQL injection attacks detection method.

In this paper, we design an effective injection attack detecting mechanism for non-relational (NoSQL) database. In our design, based on semantic structure analysis of execution statements we propose a detection approach using parse tree. While receiving an HTTP request from user, a parse tree is generated according to the user's request. Meanwhile, the old record of parse tree for the request is retrieved and used to compare with the generated parse tree. If the two trees are equal, it means that no NoSQL injection attacks involved in this request. Based on this approach, we focus on MongoDB, a popular non-relational database, to propose a dynamic NoSQL injection attacks detection mechanism called DND in the web environment. It does not require access to or modifying source codes, rewriting source codes with extra libraries, or complex assisted devices. Finally, the experimental results show that DND has high accuracy rates, low false positive rates, and low response time. It is sufficiently to demonstrate that our detection mechanism DND is efficient to counter NoSQL injection attacks for the web environment on MongoDB.

The remainder of this paper is organized as follows. In Section 2, we briefly introduce the security problems in NoSQL database. A concrete NoSQL injection attacks detection mechanism named DND proposed in Section 3. In Section 4, we present the experimental results and conclusions are drawn in Section 5.

2. **Security problems in NoSQL database.**

MongoDB and Redis are two famous and widely used NoSQL databases (http://db-engines.com/en/ranking). However, both of them have several security problems for example data files are not encrypted and also do not provide automatic encryption mechanism for data files. In order to reduce this risk, application must encrypt the data file before putting into the database. In addition, defense mechanisms are required at the operating system level such as file access permissions and file system encryption to prevent an unauthorized login.

Potential injection attacks also threaten the security of NoSQL databases. Redis and MongoDB are suffered from the risks for authentication mechanism is incomplete, lack of encryption mechanisms, and injection attacks. Here, we demonstrate an injection attack of MongoDB in Figure 1. The first statement (rows 1-3) can cause an input attack through the translation of MongoDB to ”$gt” character. The second and third statements can be inserted into the database by string concatenation. The fourth statement”$where” is a JavaScript function which is used to traverse each record in a set called myCollection. This statement cannot modify the database directly because the $where only performs read functionality. However, if an application uses this query and does not filter user's input, it will lead injection attacks.

MongoDB shell

```
db.myCollection.find(
   { a : { $gt: 3 } }
);
db.myCollection.find(
   { $where: "this.a > 3" }
);
db.myCollection.find(
   "this.a > 3"
);
db.myCollection.find(
   {$where: function() {
   return this.a > 3;}}
)
```

Figure 1. An injection attack example of MongoDB

3. Our Proposed NOSQL Injection Attacks Detection Mechanism.

3.1. NoSQL injection attack scenario.

Here, we illustrate a scenario depicted in Figure 2 to explain a procedure of NoSQL injection attack on MongoDB. Firstly, an attacker inputs an illegal URL and sends it to web server via browser. Upon receiving the request, the web server may mistake this URL is legal. Then, the web server sends this illegal URL to PHP client and the PHP client parses this URL to some code including malicious injection code. Finally, the PHP client sends it to MongDB and then MongDB executes the code. It will lead to return some information as the attacker wants or to break MongDB.

In Figure 2, we illustrate an example of data stream depicted in Figure 3. Let

Q$_1$: detected_login.php?username=Carl&password=123456

be an legal request. The data transmission of web environment is described as follows. The Q_1 can be transformed to query statement

db->logins->find(array(”username”=>$ _
POST[”username”],”password”=>$_ *POST*[”*password*”])); by PHP. This query statement in MongDB can be translated by

db.logins.find({username:’Carl’,password:’123456’}).

Let

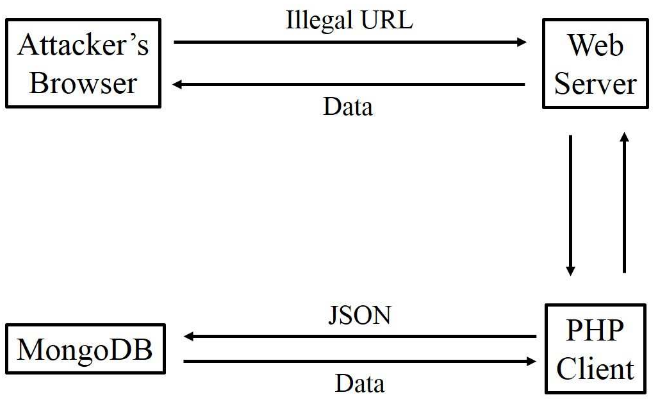


FIGURE 2. A scenario of NoSQL injection attack on MongoDB

Q$_2$: detected_login.php?username[$ne]=1&password[$ne]=1
be a malicious request. The Q$_2$ can be transformed to query statement
array("username"=>array("$ne"=> 1),"password"=>array("$ne"=> 1)).
by PHP. This query statement in MongDB can be translated by
db.logins.find(username:$ne:1,password:$ne:1).

Since $ne presents unequal in MongDB, this query statement means that to query all user names not equal to 1 and passwords not equal to 1. It is easy to see that any attacker can adopt the malicious request Q$_2$ to pass the authentication in login phase and then accesses users' information in MongDB.

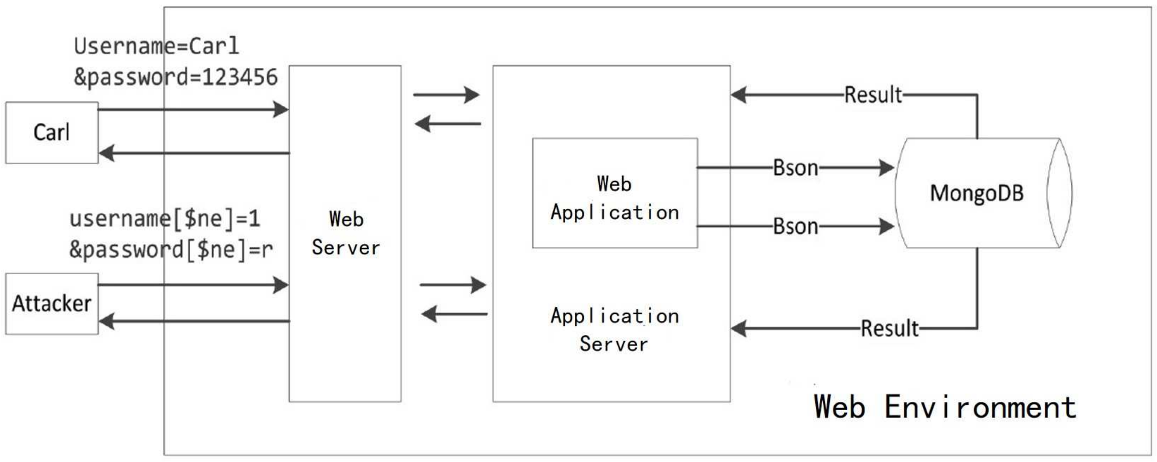


FIGURE 3. An example of data stream in Figure 2

3.2. **DND mechanism.** In our dynamic NOSQL injection attacks detection (DND) mechanism, there are eight models: Server, Modeler, Parameters Separator, Comparator, Log, MongoDB, Online judge, and Repository depicted in Figure 4.

1. **Server:** To parse Http request and obtaining parameters. Then, to splice MongoDB Shell execution statement and obtaining an execution statement called q_1. Finally, Server sends q_1 to Online judge. An example of q_1 is shown by
   http://localhost/login/login_1.php?username=CarlSun&password=
   123456;return%20true;}//

2. **Parameters Separator:** To separate the parameters of user's input in q_1 and obtaining a query statement q_2 depicted in Figure 5. Finally, Parameters Separator sends q_2 to Modeler.
3. **Modeler:** To parse q_1 into a parse tree T_1 and to parse q_2 into a parse tree T_2, respectively. Then, Modeler sends T_1 and T_2 to Comparator. The two parse trees T_1 and T_2 are shown in Figure 6 and Figure 7.
4. **Comparator:** To compare the structures of T_1 and T_2. If they are equal, then q_1 is a legal statement. Comparator sends it to MongoDB to execute. Otherwise, q_1 is viewed as an attack statement. Comparator sends this attack message to Log and q_1 to Repository.
5. **Log:** To record operations of the system, the results of Comparator, success or failures of NoSQL injection attacks detection, system running time, the time of NoSQL injection attacks occurred, and running of database.
6. **Online Judge:** To provide a quick judgement according to previous judged attack request in Repository. It can be denied access to the database directly if it is judged as an attack request. If it can't judge, to execute parameters separating, modeling, comparison, and a series of operations.
7. **Repository:** When a query statement is judged as an attack statement by Comparator, it is stored as text in Repository.

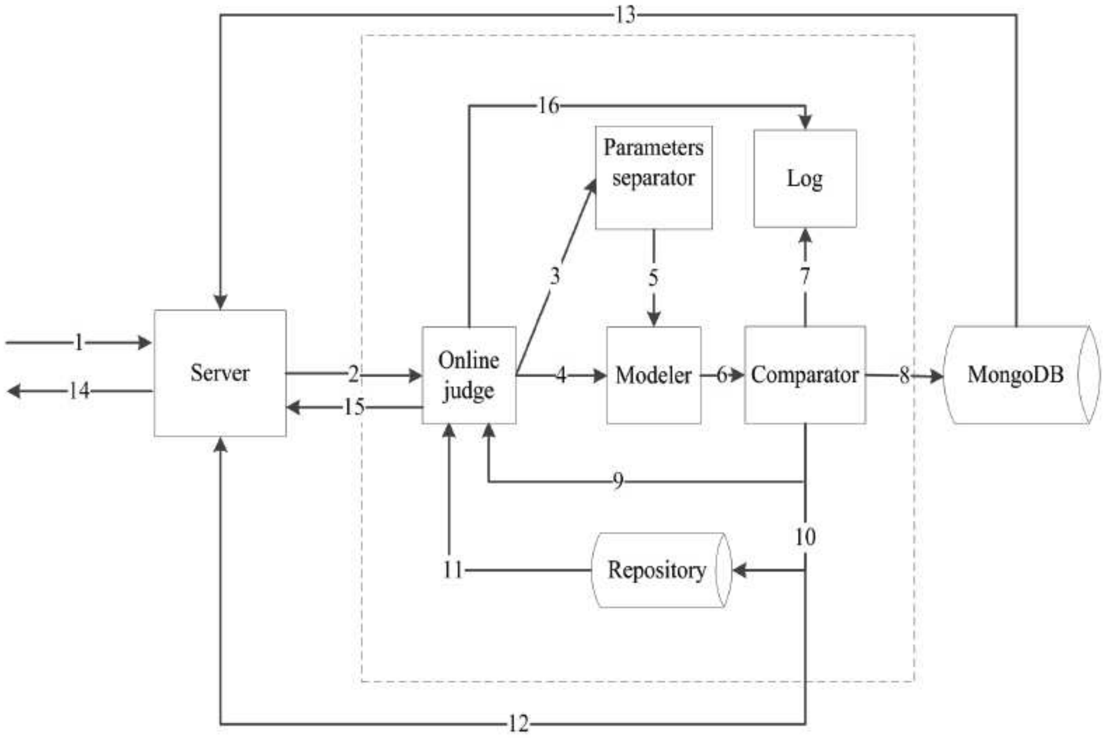


FIGURE 4. Flowchart of DND

In DND, Parameter Separator, Modeler, Comparator, and Log formed an intermediate layer. Each request from the backend application to the database is needed to pass through this layer. Our DND can protect database against NoSQL injection attacks. The detection process of DND is divided into three subprocess, which are parameters separating, modeling, and comparing. Server first sends query statement q_1 to Parameter Separator and Modeler. Then, Parameter Separator executes parameters separating for q_1 and the resulted query statement is defined as q_2. Parameter Separator sends q_2 to

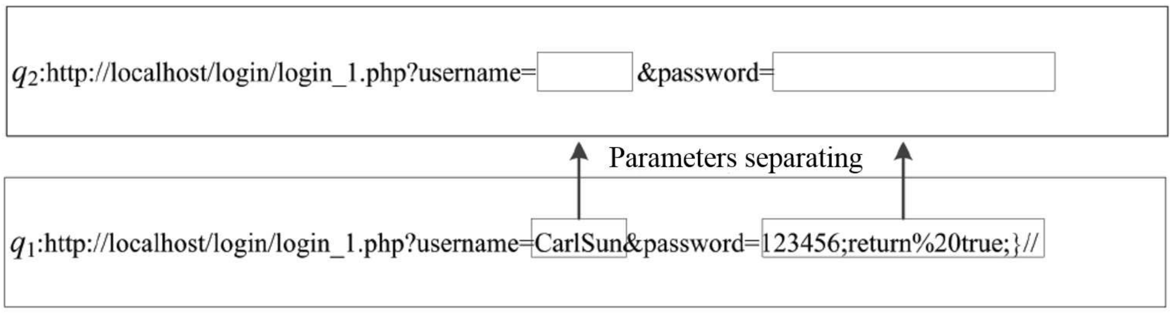

q_2:http://localhost/login/login_1.php?username= &password=
Parameters separating
q_1:http://localhost/login/login_1.php?username=CarlSun&password=123456;return%20true;}//

Figure 5. Parameters separating of q_1

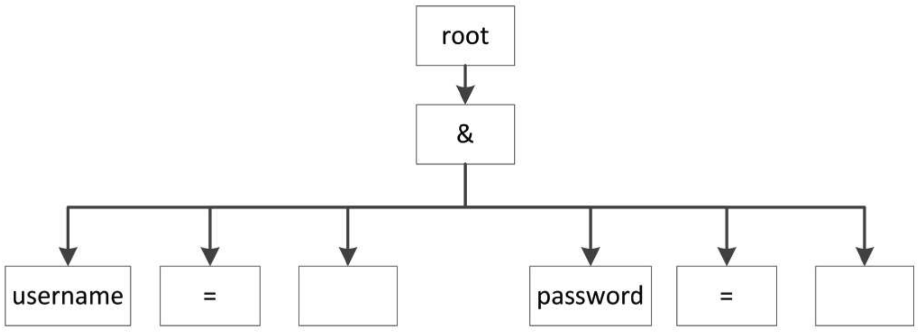


Figure 6. Parse tree T_1

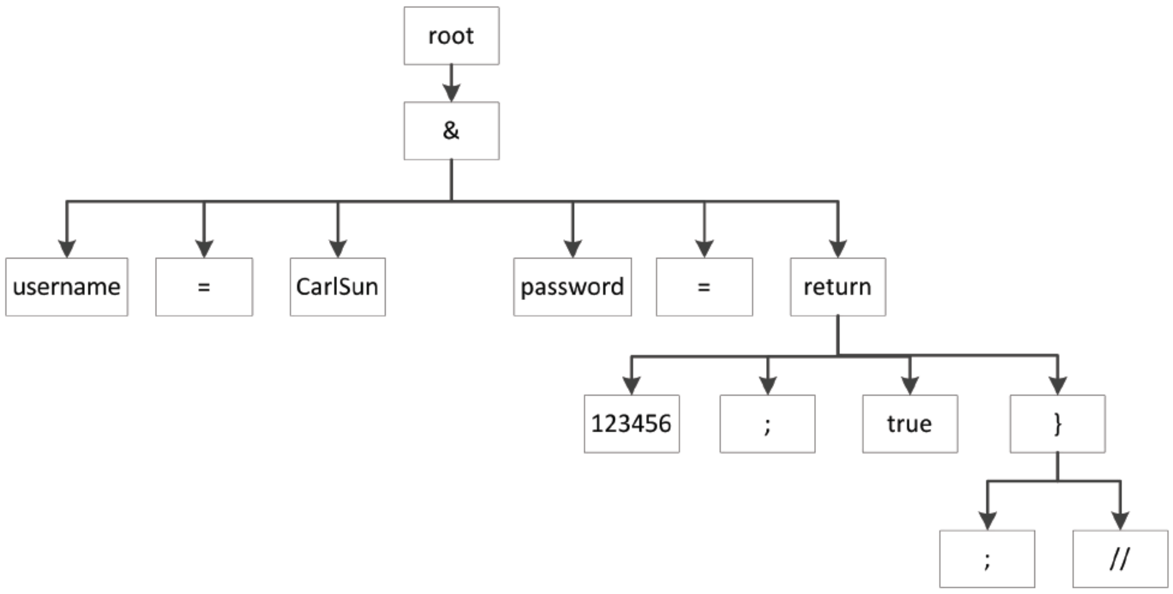


Figure 7. Parse tree T_2

Modeler. According to q_1 and q_2, Modeler generates parse trees T_1 and T_2 and sends them to Comparator. Finally, Comparator compares the structures of T_1 and T_2. If the two trees are the same, q_1 is a legal statement. Otherwise, q_1 is judged as NoSQL injection attack.

3.3. **Algorithm design.** In our DND, each MongoDB Shell statement can be parsed into a parse tree which is divided into internal nodes and leaf nodes. All internal nodes formed the structure of MongoDB Shell parse tree. When user inputs a dynamically generated MongoDB Shell statement, it will be judged as a NoSQL injection attack once the structure of parse tree is changed.

Firstly, we propose an algorithm called Extract Structure Algorithm (ESA) to demonstrate that how to construct an parse tree based on JavaScript statement and parsing it with query statement. Parse tree is a data structure to display the structure of a string. Parsing strings requires to use the relative syntax rules to partition strings. By parsing two strings and comparing the structures of the strings, the two query statements can be judged to be identical in structure.

Our ESA is described as follows. We let the query statement of the assignment symbol "=" in first layer be the first layer of parse tree. Then, splitting the left and right sides and the user's input "$_GET['ID']" is placed on the right side of the assignment symbol "=". A simple parse tree is generated shown in Figure 8. The key point is to classify characters. Firstly, the assignment statement is formed to be a character set at the first level. Then, the JavaScript identifier is formed to be a character set at the second level. Finally, value is formed to be a character set at the third level.

If attacker inputs "12;return true;//" in $q.="if(this.id==id)return true;", then injection attack $q statement can be constructed to a parse tree T_3, where operators "=", ";", "//" are first divided and then the JavaScript's keyword "var", "return", "if" are second divided. The parse tree T_3 is depicted in Figure 9.

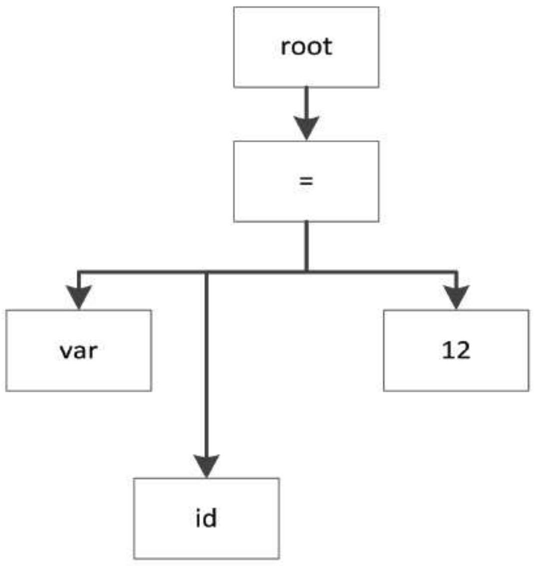


FIGURE 8. A simple parse tree

Then, we propose another algorithm called verification model algorithm (VMA) which is used to compare two parse trees' structures. Here, we parse JavaScript statement $q as $q="var id=".$_GET['id'].";"; and $q.="if(this.id==id)return true;". If id=12, it formed a parse tree T_4 shown in Figure 10. It is easy to see that T_3 and T_4 have different structures. We can compare the two structures to judge a query statement whether is a NoSQL injection statement.

Finally, we adopt Trie tree algorithm to present data structure in Online judge. For example, the following attack statements can be transformed to Trie tree shown in Figure 11.

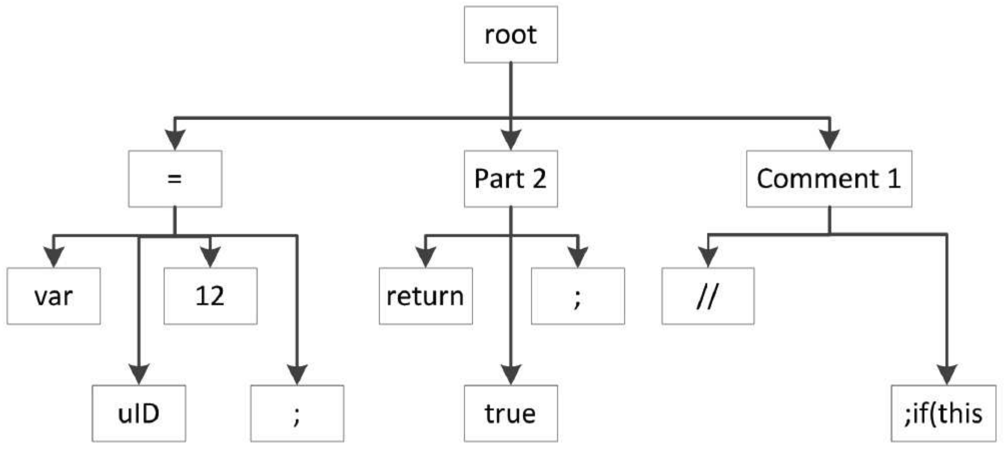


FIGURE 9. Parse tree T_3

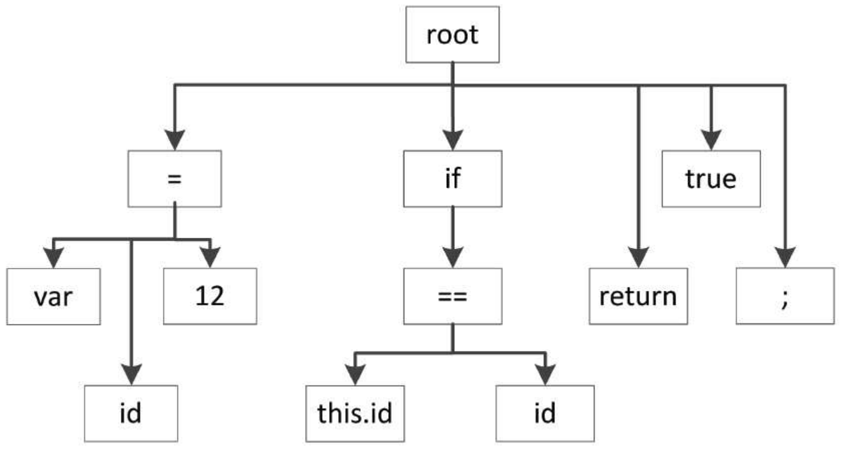


FIGURE 10. Parse tree T_4

4. **Experimental Results.** In this section, we demonstrate several experimental results of our DND in terms of accuracy, false positive, and efficiency.

4.1. **Environment and test set.** Our experimential environment shown in Figure 12 is based on two same PCs (PC$_A$ and PC$_B$) with processor Intel(R)Core(TM)I5-2400CPU @3.10GHz, 8GB memory, 1T Hard disk, and the operating system is Windows 7. Meanwhile, we deploy Nginx as web server, MongoDB, and DND in PC$_B$. Note that DND is deployed on web server and the framework is depicted in Figure 13.

Since NoSQL injection attacks are novel attack approaches in NoSQL database, there are no open test sets for testing. In order to test the effectivity of our DND, we first design four types of web applications summarized in Table 1. Then, we generate five thousand attack samples as test set which is randomly generated by existed NoSQL injection attack statements. Our test set has covered all types of NoSQL injection attacks and open source in Github (https://github.com/youngyangyang04/NoSQLInjectionAttackDemo/tree /master/attackSet).

4.2. **Results.** In Table 2, we demonstrate the detecting accuracy rate of DND. It is easy to see that our DND has very high detecting accuracy rate in four different web applications. In Table 3, we show the false positive rate of DND. Obviously, our DND

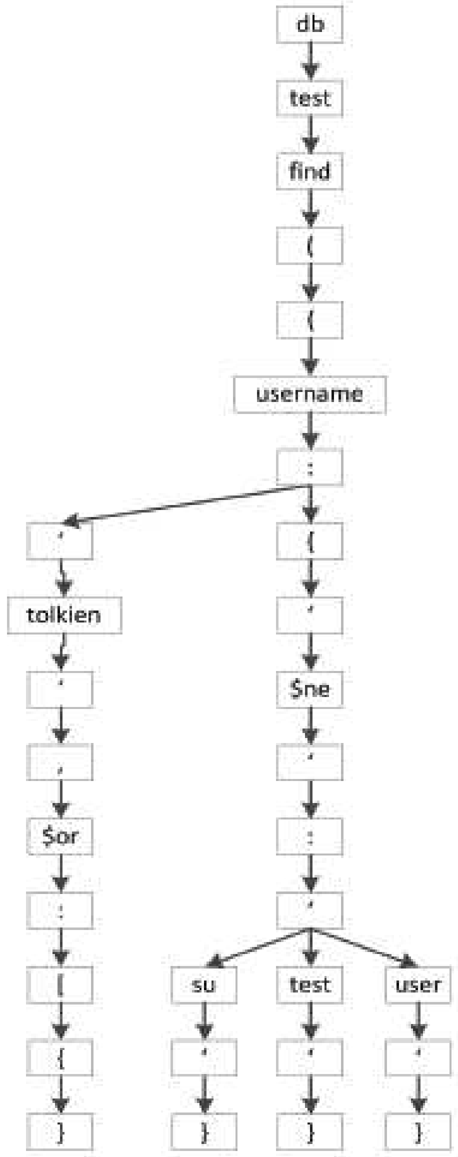


Figure 11. An example of Trie tree

has a very low false positive rate. Finally, we demonstrate the performance comparisons of running four applications with DND in Figure 14. Note that each black rectangle presents the average runtime of single query statement without NDN and each white rectangle presents the average runtime of single query statement with NDN. It is easy to see that our DND has better performance and can be used to prevent NoSQL injection attacks.

5. **Conclusions.** In this paper, we have proposed a dynamic NoSQL injection attacks detection mechanism called DND in the web environment. It does not require access to or modifying source codes, rewriting source codes with extra libraries, or complex assisted devices. According to the experiments, DND has high accuracy rates, low false positive rates, and low response time. It is sufficiently to demonstrate that our detection mechanism DND is efficient to counter NoSQL injection attacks for the web environment on MongoDB.

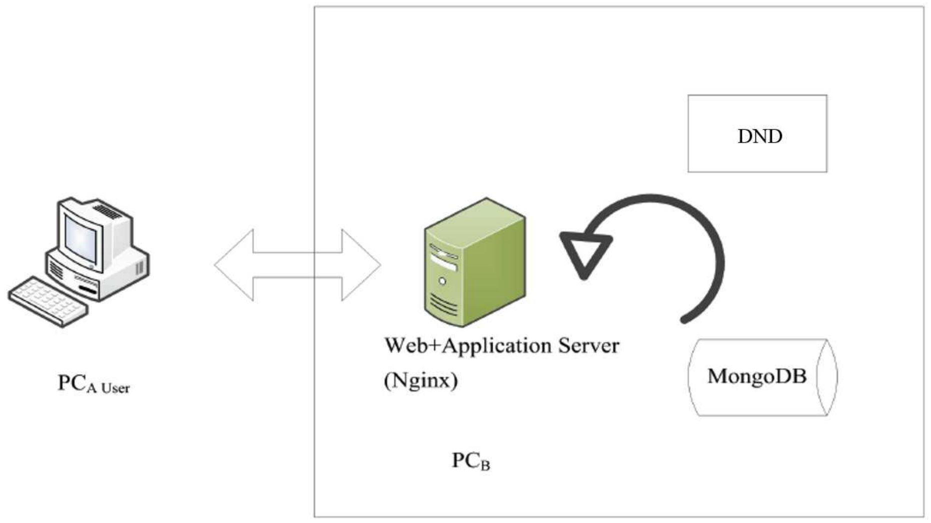


Figure 12. Experimental environment

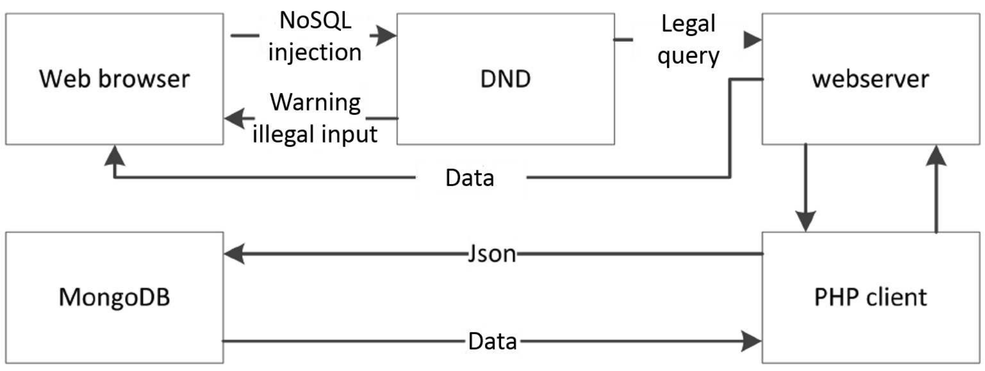


Figure 13. The framework of DND deployed on web server

Our DND is only focused on MongoDB. There are several non relational databases such as Redis and Cassandra have been used by companies and enterprises. These databases also exist the risk of injection attacks. Currently, due to NoSQL injection attacks are novel attack approaches, the number of attack methods are very less. However, more and more attack approaches will be found in the future.

Acknowledgment. The authors would thank anonymous referees for a valuable comments and suggestions. This work is supported by the Science and Technology Project of Fuzhou under Grant No.2017-G-7.

REFERENCES

[1] R. Cattell, Scalable sql and nosql data stores, *Acm Sigmod Record*, vol. 39, no. 4, pp. 12–27, 2011.
[2] E. Sahafizadeh and M. A. Nematbakhsh, A survey on security issues in big data and nosql, *Advances in Computer Science: an International Journal*, vol. 4, no. 4, pp. 68–72, 2015.
[3] M. Stonebraker, Sql databases v. nosql databases, *Communications of the ACM*, vol. 53, no. 4, pp. 10–11, 2010.
[4] A. K. Zaki, Nosql databases: new millennium database for big data, big users, cloud computing and its security challenges, *International Journal of Research in Engineering and Technology (IJRET)*, vol. 3, no. 15, pp. 403–409, 2014.
[5] R. F. Puppy, Nt web technology vulnerabilities, *Phrack Magazine*, vol. 8, no. 54, 1998.

Table 1. Our designed web applications

Applications	Language	Functionality	Attacked results
Information-get	PHP	Returning the specified information	Returning all information of specific collection in database
Information-march	PHP	Matching information for specific information	Matching information for any input
MongoDB-register	JavaScript	User registration	Injecting dirty data into the collection of database and using it to perform illegal operations
MongoDB-admin	JavaScript	User login	User login is performed without password

Table 2. Detecting accuracy rate of DND

Applications	Number of attack statements	Faults	Accuracy (%)
Information-get	2000	0	100%
Information-march	1000	0	100%
MongoDB-register	1000	0	100%
MongoDB-admin	1000	0	100%

Table 3. False positive rate of DND

Applications	Number of attack statements	False positive	False rate (%)
Information-get	2000	0	0
Information-march	1000	0	0
MongoDB-register	1000	0	0
MongoDB-admin	1000	0	0

[6] S. Roy, A. K. Singh, and A. S. Sairam, Detecting and defeating sql injection attacks, *International Journal of Information and Electronics Engineering*, vol. 1, no. 1, p. 38, 2011.

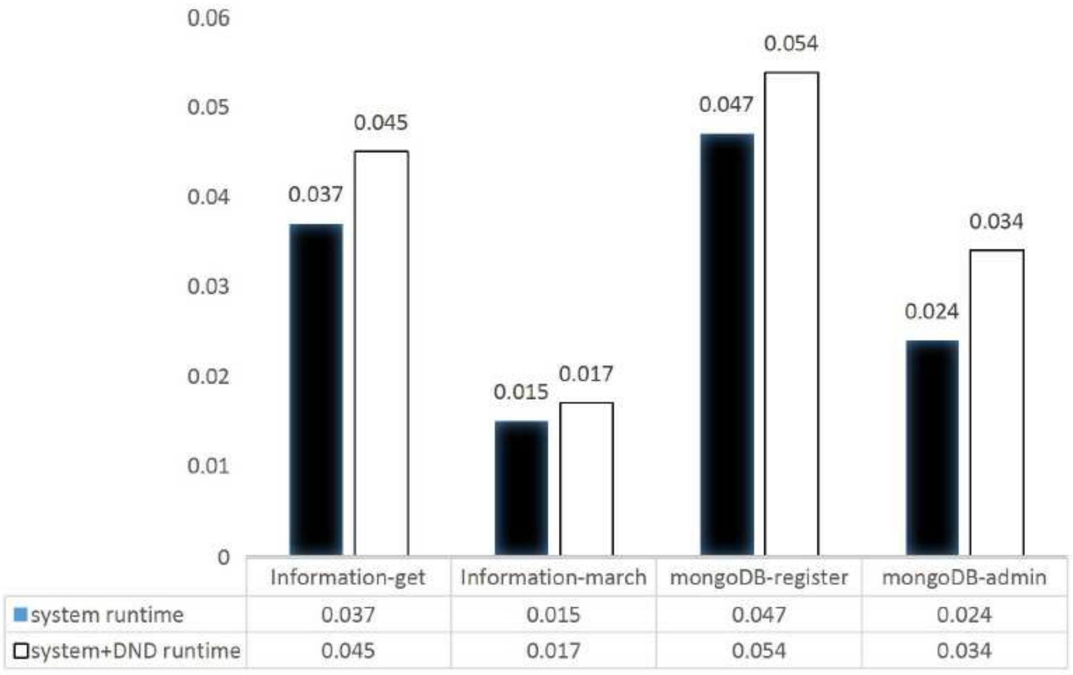
[7] D. Henderson, M. Lapke, and C. Garcia, Sql injection: A demonstration and implications for accounting students, *AIS Educator Journal*, vol. 11, no. 1, pp. 1–8, 2016.

[8] E. Pearson and C. L. Bethel, A design review: Concepts for mitigating sql injection attacks, in *Digital Forensic and Security (ISDFS), 2016 4th International Symposium on*, pp. 169–169, IEEE, 2016.

[9] L. Liu, J. Xu, H. Yang, C. Guo, J. Kang, S. Xu, B. Zhang, and G. Si, An effective penetration test approach based on feature matrix for exposing sql injection vulnerability, in *Computer Software and Applications Conference (COMPSAC), 2016 IEEE 40th Annual*, vol. 1, pp. 123–132, IEEE, 2016.

[10] C. Gould, Z. Su, and P. Devanbu, Jdbc checker: A static analysis tool for sql/jdbc applications, in *Proceedings of the 26th International Conference on Software Engineering*, pp. 697–698, IEEE Computer Society, 2004.

[11] M. Bravenboer, E. Dolstra, and E. Visser, Preventing injection attacks with syntax embeddings, in *Proceedings of the 6th international conference on Generative programming and component engineering*, pp. 3–12, ACM, 2007.



	Information-get	Information-march	mongoDB-register	mongoDB-admin
■system runtime	0.037	0.015	0.047	0.024
□system+DND runtime	0.045	0.017	0.054	0.034

Figure 14. The efficiency comparisons

[12] X. Fu, X. Lu, B. Peltsverger, S. Chen, K. Qian, and L. Tao, A static analysis framework for detecting sql injection vulnerabilities, in *Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International*, vol. 1, pp. 87–96, IEEE, 2007.

[13] Y. Kosuga, K. Kono, M. Hanaoka, M. Hishiyama, and Y. Takahama, Sania: Syntactic and semantic analysis for automated testing against sql injection, in *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, pp. 107–117, IEEE, 2007.

[14] W. G. Halfond and A. Orso, Amnesia: analysis and monitoring for neutralizing sql-injection attacks, in *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, pp. 174–183, ACM, 2005.

[15] I. Lee, S. Jeong, S. Yeo, and J. Moon, A novel method for sql injection attack detection based on removing sql query attribute values, *Mathematical and Computer Modelling*, vol. 55, no. 1, pp. 58–68, 2012.

[16] Y.-W. Huang, S.-K. Huang, T.-P. Lin, and C.-H. Tsai, Web application security assessment by fault injection and behavior monitoring, in *Proceedings of the 12th international conference on World Wide Web*, pp. 148–159, ACM, 2003.

[17] F. Valeur, D. Mutz, and G. Vigna, A learning-based approach to the detection of sql attacks, in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pp. 123–140, Springer, 2005.

[18] R. Komiya, I. Paik, and M. Hisada, Classification of malicious web code by machine learning, in *Awareness Science and Technology (iCAST), 2011 3rd International Conference on*, pp. 406–411, IEEE, 2011.

[19] S. W. Boyd and A. D. Keromytis, Sqlrand: Preventing sql injection attacks, in *International Conference on Applied Cryptography and Network Security*, pp. 292–302, Springer, 2004.

[20] W. Halfond, A. Orso, and 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.

[21] D. Mitropoulos and D. Spinellis, Sdriver: Location-specific signatures prevent sql injection attacks, *computers & security*, vol. 28, no. 3, pp. 121–129, 2009.

[22] P. Bisht, P. Madhusudan, and V. Venkatakrishnan, Candid: Dynamic candidate evaluations for automatic prevention of sql injection attacks, *ACM Transactions on Information and System Security (TISSEC)*, vol. 13, no. 2, p. 14, 2010.

[23] G. Buehrer, B. W. Weide, and P. A. Sivilotti, Using parse tree validation to prevent sql injection attacks, in *Proceedings of the 5th international workshop on Software engineering and middleware*, pp. 106–113, ACM, 2005.

[24] T.-Y. Wu, J.-S. Pan, C.-M. Chen, and C.-W. Lin, Towards sql injection attacks detection mechanism using parse tree, in *Genetic and Evolutionary Computing*, pp. 371–380, Springer, 2015.

[25] T.-Y. Wu, C.-M. Chen, X. Sun, S. Liu, and J. C.-W. Lin, A countermeasure to sql injection attack for cloud environment, *Wireless Personal Communications*, pp. Doi: 10.1007/s11277–016–3741–7.

[26] A. Ron, A. Shulman-Peleg, and E. Bronshtein, No sql, no injection? examining nosql security, *arXiv preprint arXiv:1506.04082*, 2015.

[27] L. Okman, N. Gal-Oz, Y. Gonen, E. Gudes, and J. Abramov, Security issues in nosql databases, in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on*, pp. 541–547, IEEE, 2011.