MongoDB NoSQL Injection Analysis and Detection

Boyu Hou
Department of CS
Kennesaw State University
Marietta, GA, USA
bhou@students.kennesaw.edu

Kai Qian, Lei Li, Yong Shi
Department of CS
Kennesaw State University
Marietta, GA, USA
{kqian, lli13, yshi5}@kennesaw.edu

Lixin Tao
Department of CS
Pace University
New York, USA
ltao@pace.edu

Jigang Liu
Department of ICS
Metropolitan State University
St. Paul, MN, USA
jigang.liu@metrostate.edu

***Abstract*— A NoSQL, also called a "Non-Relational" or "Not only SQL," database system provides an approach to data management and database design for very large sets of distributed data and real-time web applications. A NoSQL database system is also a popular data storage for information retrieval because it supports better scalability, availability, and faster data access while comparing with traditional relational database management systems (RDBMS). What the RDBMS data needs is predictable as its data is stored in structured tables by defining the relationship between the different columns. In contrary the data in NoSQL databases does not need to be stored in a structured or fixed fashion. When performance and real-time access are more concerned than consistency, such as indexing and retrieving large numbers of records, NoSQL databases are more suitable than relational databases. With their obvious advantages in better performance, scalability, and flexibility, NoSQL databases have been adopted lately by many small businesses as they are moving their increasing business data into the clouds. However, the research on the security of a specific NoSQL database system or NoSQL database systems in general is very limited. Although there are many storage advantages in NoSQL databases, the need of quick and easy access to data has been seriously affected by the security issue of NoSQL databases. This paper examines the maturity of security measures for MongoDB, a typical NoSQL database system, with aspects in both attack and defense at the code level. The experimental testing on NoSQL injections is performed with JavaScript and PHP. After the demonstration on how a server-side JavaScript injection attack against a NoSQL database system reveals the customer's private data, two methods are discussed in preventing this type of security problems from happening. It is believed that our study will help database developers not only realizing that NoSQL database systems are not designed with security as a priority but also learning how to build a security layer to their organizations' NoSQL applications to avoid NoSQL injections.**

Keywords—NoSQL Injection, Security Analysis, Defense, Detection Level

I. INTRODUCTION

Database security has been and will continue to be one of the most critical aspects of application security. Access to a database grants an attacker a dangerous amount of control over the most critical information. For example, SQL injection is a code injection technique that is used to attack data driven applications, in which malicious SQL statements are inserted into an entry field for execution. The malicious user can use SQL commands insert to the Web form submission or enter the domain name to achieve the purpose of tricking the server to execute malicious SQL commands.

Recently, the NoSQL databases have become more and more popular because NoSQL databases provide looser consistency restrictions than traditional SQL databases do. By requiring fewer relational constraints and consistency checks, NoSQL databases often offer performance and scaling benefits [1]. Some examples are MongoDB [2], and Cassandra [3]. MongoDB can be adapted to all sizes of businesses and individuals since it is open source database. The data pattern can be updated flexibly with the development of the application while providing a secondary index and complete inquiry system. But does that mean NoSQL database systems are immune to injections? As an alternative to traditional relational database, NoSQL is a broad class of database management systems that are not traditional relational database management systems. They do not use SQL language as the primary query language, nor do they typically require fixed table schemas. NoSQL database system allows a user to change data attributes at any time, and data can be added anywhere. There is no need to specify a document design or even a collection up-font. NoSQL databases pay more attention to real-time data processing capability and they are good at directly operation on data access, which greatly promote the development of interactive software system. One of the biggest advantages is the ability to change attributes because of the weakening of structural, so modification process is very convenient.

However, this big advantage also affects its safety where the highest incidence of attacks is injection attacks. It has also been ranked in public enemy number one in the field of database relations. The face that NoSQL does not use SQL language in queries does not mean it is immune to the threat of injection attacks. Many claim that SQL injection does not

978-1-5090-0946-6/16 $31.00 © 2016 IEEE
DOI 10.1109/CSCloud.2016.57

work in NoSQL, but the principle is exactly the same: all the attacker would need to do is change the grammar form of injection content. That is to say, although SQL injection will not happen, JavaScript injection or JSON [4] injection can also threaten the security. Our study shows that these databases are still potentially vulnerable to injection attacks, even if they do not use the traditional SQL syntax, because these NoSQL injection attacks may be executed within a procedural language, rather than in the declarative SQL language such as PHP injection attack and arbitrary JavaScript injection. In this paper we hope to raise the awareness of developers and information security owners to NoSQL security with our examples in both injection and protection.

II. Related Works

Many database security threats are caused by database vulnerability. As for injection, there are many methods to learn. Database type, version, and other information are used to identify the motivation for this type of attack. One purpose is to collect the type and structure of the database to prepare for other types of attacks, which can be described as a preparatory step attack. It can be injected by using JavaScript and PHP code [5]. Query in database is usually using input method, which means that there will be variables or parameters to store the value and pass it. To exploit the parameters, one can use HTTP URL as a way for injecting which contains GET or POST [6]. This paper tries to find vulnerability and inject.

III. NOSQL SECURITY THREAT AND INJECTION

We are living in an information age with a large amount of information generated every day. Therefore, it is very important to record, assign and manage these big data. Once the data leak, persons, companies, even the entire society would suffer enormous losses which lead developers to spend more time on database security. NoSQL database gradually becomes the main database for personal use and companies' management. However, the methods that hackers use to attack SQL databases can also be used for attacking NoSQL databases, thus NoSQL database security threat and detection problems are crucial and increase every day.

A. MongoDB NoSQL Database Security Threat

In general, databases provide the functions for creating, reading, updating and deleting the data. Among them, query is the function we use more often compared to others especially in the library systems. As for query, database systems would provide an input box to allow users to input information of the book they want to find. When developers are developing the system, they will store the value from input boxes into variables and these variables will be used in query statements. The variables might be letters, numbers or a sentence containing notations which will be all sent to query statements. These variables will be used in execution of query. This will bring a very dangerous case, if developers do not check the values or variables. The threat is that the malicious code will be executed when the query function is running which produces an extremely negative influence upon database security. The malicious code will disturb the condition statements, for example, it will let the result always be true by using notations and segmentations in SQL statement. It is also the same when it is used in NoSQL MongoDB. The query statement of SQL and NoSQL MongoDB is shown in Fig.1 and we use query books by ISBN numbers as an example.

```
SQL: "SELECT * FROM users WHERE (ISBN =
'" + isbn_number + "');"

MongoDB: db.collection.find( { ISBN:
isbn_number } )
```

Fig. 1 Permissions for killing process

As we can see, hackers may write malicious codes and input them into the input boxes in order to be executed, which is called injection. Because the malicious code can be directly put into condition statements or other statements, hackers are able to gain information by injection successfully. Injection is a very annoying security vulnerability which all web, software and system developers should pay attention to. No matter what platform, technology or data layer, we need to make sure they understand and prevent vulnerability. Unfortunately, developers often do not take the time to concentrate on it, as well as their applications, and their customers are extremely vulnerable to be attacked.

B. MongoDB NoSQL Database Injection

MongoDB is a cross-platform document-oriented database which belongs to NoSQL databases and has been used more and more in personal applications and enterprise management. We will take MongoDB as an example of injection and defense, detection and analysis for NoSQL database.

Injection is a common method to attack databases. Injection attacks database by inputting (injecting) a sentence of malicious code in order to let it be inserted to execute code from application or software. By using this, it can obtain user authority or root authority, obtain information from database, update or delete data into database, etc. The worst case is deleting data, because it will produce an extremely negative influence upon person or company. Basically, hackers first need to find the position in order to let the malicious code in; Second, they have to figure out the type of the database victim uses; Third, malicious code has to be programmed for injection. In general, injection for NoSQL database is the same as injection in SQL database. Hackers inject malicious code into input boxes or directly into URL. As for search functions, the malicious code becomes a variable that participates in execution. Under normal conditions, program starts to search based on what users input. It will search row by row and output the current result. When malicious code is injected, the result of that sentence will be always true, so that program will output all the data without current input.

Assuming there is a library system using MongoDB database which stores all the book information such as name, author, ISBN number, publisher, etc. Users search the book by inputting ISBN numbers in input box and if the number is matched, the system will show the details of the current book. Current output with ISBN number 0763754891 is shown in Fig.2.

```
Array ( [_id] => MongoId Object ( [$id] =>
5654b9b44126f790e3f03222 ) [ISBN] => 0763754891
[Title] => Web Development with JavaScript and AJAX
Illumination [Author] => Richard Allen, Kai Qian , LiXin
Tao, Xiang Fu [Publisher] => Jones& Bartlett [Date] =>
2009 ) 1
```

Fig. 2 Current query result

We mainly introduce three injection ways: one is to inject directly by input boxes, the other is to inject by URL. First, assuming the system is written by PHP and JavaScript for passing value and querying which is shown in Fig.3.

```
$unsearch = $_POST['name'];
$jss = "function() {return this.ISBN == '$unsearch';}";
$cursor = $users->find(array('$where' => $jss));
```

Fig. 3 PHP and JavaScript for passing value and querying

In this case, the system receives the value from input box by `$_POST['name']`, and then sends it to a piece of JavaScript `$jss`. Next, the system starts to query by the feature of ISBN number: `return this.ISBN`. At this time, the system starts to search document by document and output the book information which is matched by `find(array('$where' => $jss))`. It means if the result of the `find()` condition statement is true, it will output the book information. Thus, the hackers could inject a piece of malicious code to make it always true so that all the answers will be true while searching document by document. The malicious code could be like this: 0763754891' || '1' =='1. By using this, no matter what the hackers input, because of the || '1' =='1, the result of this condition statement is always true so that it will output all the data in this database.

The second way is to inject by URL. Assuming the system is written by PHP and after the user clicks submit or search, the query starts. At this time, the URL of next page will be similar to: XXX/XXX.php?isbn=XXX. When the ISBN number is matched, it will show the current book information. The system also searches for book information by the given feature. The code is similar to the one shown in Fig.4.

```
$search = array (
            'ISBN' => $_GET['isbn']
        );
$cursor = $users->find($search);
```

Fig. 4 PHP for querying

In the meantime, ScarePackage has a boot receiver to run the application during the device starting period. The necessary permissions are shown in Fig. 5. In this case, the system receives the value by `$_GET['isbn']` and passes it to `array`. Based on the MongoDB query method of Specify Equality Condition: `db.collection.find( { ISBN : "XXX" } )`, it will output the book information which is matched as long as the condition statement is true. That means if the hackers can make the answer of the condition statement always true, it will output all the information without inputting the current ISBN number. The current input normally is ISBN=0763754891 which is true, but on the other hand, ISBN!=123456 is also true. As long as the number is not the ISBN number, the condition statement will be true, which means the probability is pretty high. According to the parameter standard, we can use [$ne] in URL instead of '!' [7]. Hence, the injection URL could be XXX/XXXphp?isbn[$ne]=1. The injection result is shown in Fig.5.

```
Array ( [_id] => MongoId Object ( [$id] =>
5654b9b44126f790e3f03222 ) [ISBN] => 0763754891 [Title]
=> Web Development with JavaScript and AJAX Illumination
[Author] => Richard Allen, Kai Qian , LiXin Tao, Xiang Fu
[Publisher] => Jones& Bartlett [Date] => 2009 ) 1
Array ( [_id] => MongoId Object ( [$id] =>
5654b9b44126f790e3f03223 ) [ISBN] => 9780763754204
[Title] => Software Architecture and Design Illuminated
[Author] => Kai Qian, Xiang Fu, LiXin Tao, Jorge Diaz-
Herrera,Chong-wei Xu [Publisher] => Jones & Bartlett [Date]
=> 2009 ) 1
Array ( [_id] => MongoId Object ( [$id] =>
5654b9b44126f790e3f03224 ) [ISBN] => 9780763754204
[Title] => Software Architecture and Design Illuminated
[Author] => Kai Qian, Xiang Fu, LiXin Tao, Jorge Diaz-
Herrera,Chong-wei Xu [Publisher] => Jones & Bartlett [Date]
=> 2009 ) 1
```

Fig. 5 Some of the books information in database

By using this strategy, because the MongoDB execution also accepts notations such as $, injection can also be executed in shell, which is shown in Fig.6.

```
db.testCollection.find({ISBN:{$ne:'1'}})
 "_id" : ObjectId("5654b9b44126f790e3f03222"), "
"Web Development with JavaScript and AJAX Illumi
en, Kai Qian , LiXin Tao, Xiang Fu", "Publisher"
009 }
 "_id" : ObjectId("5654b9b44126f790e3f03223"), "
 : "Software Architecture and Design Illuminated
u, LiXin Tao, Jorge Diaz-Herrera,Chong-wei Xu",
, "Date" : 2009 }
 "_id" : ObjectId("5654b9b44126f790e3f03224"), "
 : "Software Architecture and Design Illuminated
u, LiXin Tao, Jorge Diaz-Herrera,Chong-wei Xu",
, "Date" : 2009 }
```

Fig. 6 Some of the books information in shell

Basically, as for injections in query, the main idea is to make the condition statement always true so that the result can

get passed by searching document by document and shows hackers all the information in database.

IV. MONGODB NOSQL DEFENSE AND DETECTION ANALYSIS

Injection could cause the leak of personal account and company information. The hackers can obtain root authorization or exactly what they pursue by injection which produces an extremely negative influence upon information security. Thus, it is necessary to add the function of defense in order to improve the system security. As for the defense, here we mainly discuss how to avoid injection from input boxes in websites. And then, according to the vulnerability and some features, we will provide a detection approach for threat level.

A. MongoDB NoSQL Defense Analysis

Basically, for MongoDB defense, there are two ways: input validation and parameterized statement. The first one, input validation, is to limit what users input. As for ISBN number query, it will be all numbers. Thus, when developers are building the system or software, they can add a piece of JavaScript code to limit the input boxes: let it only accept numbers, which is shown in Fig.7.

```
onkeypress="return
event.keyCode>=48&&event.keyCode<=57"
```

Fig. 7 JavaScript code for limiting input

Since the malicious code usually contains notations, this solution can prevent variables from passing unsafe characters. Furthermore, developers can add JavaScript code to let input boxes only accept letters or combinations such as numbers + letters. This solution is to check the input before it passed to variables which are easy to add to system or software.

The second one, parameterized statement, is to check and filter the variables. At the time of writing condition statements, variable for user input is not directly embedded into the condition statement. That is, the user's input absolutely cannot be embedded directly into the condition statement. Instead, the contents of the user's input must be filtered, or use parameterized statements to pass variables entered by the user. Parameterized statements use parameters instead of embedding user input variable into the condition statement. With this measure, we can eliminate most of injection attacks. For example, there is a piece of code to determine the characters in a variable if it contains numbers only or not. By checking the ISBN number, if yes, pass the value; if not, reject. The code is shown in Fig.8.

```
if(is_numeric($unsearchtwo)=="ture"){}
else echo "Incorrect.";
```

Fig. 8 Parameterized statement code

Besides, developers could also check and filter other features such as notations, space or some specific characters. Meanwhile, systems are supposed to have many layers to make sure the values users input cannot be directly passed into condition statement. As long as the malicious code cannot be executed, the data will be much safer.

B. Malicious Feature Detection

Malicious feature detection is to detect if the system or software has some features which are dangerous for security. Based on some malicious code and features, we provide the diagram in Fig.8.

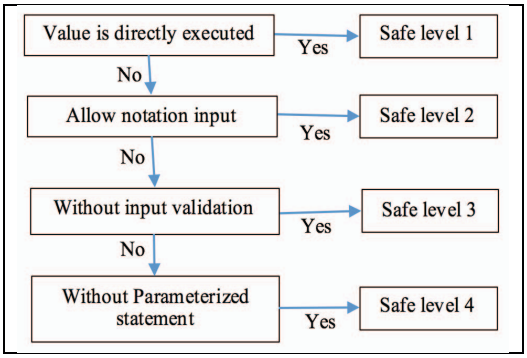

Fig. 9 JavaScript code for limiting input

This detection can help developers detect their safe level in their works. The higher the safe level number is, the more secure the NoSQL database is.

V. CONCLUSION AND FUTURE WORK

In general, developers should apply defense methods to their NoSQL database systems to prevent injections or any other attacks from happening. In addition, security layers are recommended to be built to keep malicious code away from the systems.

In the future work, we plan to examine new injection possibilities and other vulnerabilities particularly on NoSQL databases, platforms and languages, as well as to study how to defend and mitigate attacks so that NoSQL databases are safe to use.

REFERENCES

[1] https://www.owasp.org/index.php/Testing_for_NoSQL_injection

[2] https://www.mongodb.org/

[3] http://cassandra.apache.org/

[4] https://securityintelligence.com/does-nosql-equal-no-injection/

[5] http://software-talk.org/blog/2015/02/mongodb-nosql-injection-security/

[6] http://blog.trendmicro.com/trendlabs-security-intelligence/zero-day-vulnerability-found-in-mongodb-administration-tool-phpmoadmin/#

[7] http://blog.imperva.com/2014/10/nosql-ssji-authentication-bypass.html