JKAU: Comp. IT. Sci., **Vol**. **8** No. 1, pp: 85 – 93 (1440 A.H. / 2019 A.D.)
Doi: 10.4197/Comp. 8-1.8

Security Testing Tool for NoSQL Systems

Muhammad A. Lawal and **Mostaf A. Saleh**

Department of Computer Science, Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah, Saudi Arabia

mlawal@stu.kau.edu.sa

Abstract. NoSQL systems are becoming more popular due to their inherent advantages and solutions it provides to the limits of a relational database. However, despite its benefits, it comes with security challenges. In this paper, an input validation mechanism architecture is proposed for Mongo DB to detect and prevent NoSQL injection attacks, the mechanism employs a Deterministic Finite Automaton (DFA) approach to detect and prevent attacks on NoSQL systems. Furthermore, a security comparison of some NoSQL systems is provided based on recent literature. The security features compared are authentication, authorization, data encryption and input validation. The proposed mechanism will improve the security of Mongo DB system because invalid inputs requests will be detected and prevented from being processed.

Keywords: NoSQL, MongoDB, Deterministic Finite Automaton, Security testing.

1. Introduction

The emergence of the big data has strained the capacity of the relational database management system and makes it challenging to deal with these data because of some of its properties. The acceptance of the social networking and the pursuit of research findings is the cause of the majority of the data generated, which accounts for over 75% of the data generated in the previous decade. Facebook and subatomic particles-collision experiments only generate terabytes of data on average per day and per minutes respectively [1]. On the other hand, the advent of the NoSQL systems has provided solutions to the problems of dealing with such enormous data.

NoSQL systems in contrast to the relational database management systems offer a different paradigm for storing data. It handles unstructured data such as multimedia, emails, documents e.t.c efficiently. The general features of NoSQL systems can be summed up as high scalability, reliability, and very simple data model although prior systems lack support for security at the database level [2].

Recently, newer systems have built-in security mechanisms [3]. However, efforts to develop more secure systems are still needed especially at the application entry point, which provides an opportunity for interaction between the user and the database. A user could manipulate inputs to perform injection attack or get unauthorized access to a system. Hence the need for strong input validation schemes is imperative.

An input validation mechanism architecture using deterministic finite automata (DFA) is proposed for Mongo DB to detect and prevent NoSQL injection attacks. In

addition, security comparison of some NoSQL systems is provided based on recent literature. The security features compared are authentication, authorization, data encryption and input validation. Authentication is the process of confirming if a user is who he claims to be, while authorization relates to verifying if the authenticated user is permitted to do a particular action[4]. Input validation ensures that the data is in right syntax, within length boundaries, consists only allowable characters, or that numbers are properly signed and in range boundaries [5].

The remainder of the paper is organized as follows: Section 2 presents the background and types of NoSQL systems as well as the comparison, Section 3 presents the related work. The proposed architecture is described in section 4 and finally, section 5 concludes the paper.

2. Background

NoSQL stands for "Not Only SQL" and the word was used for the first time in 1988 [6] and was re-introduced in 2009 in an event concerning distributed databases. The event deliberated on the new technologies being presented by Google (Google Big Table [7]) and Amazon (Dynamo [8]) to manage huge amounts of data. Since then, attention in the research of NoSQL technologies flourished, and central to several publications.

NoSQL databases deliver an all-inclusive solution for a broad range of database concerns, with features: basically available, soft state, eventually consistent (BASE) instead of relational models of atomicity, consistency, isolation, and durability (ACID) [9].NoSQL provides a faster rate of data processing. It also provides a moderately low-cost for enterprises to efficiently manage big volumes of data [10]. NoSQL can widely be classified into four categories namely Key-value database, Column-oriented database, Document-based and Graph database.

A. Types of NoSQL Databases

NoSQL databases are classified as Key-value database (Redis), Column-oriented database (HBase), Document-oriented based (Mongo DB) and Graph database (Neo4j). This classification is based on the data model. Below is a brief description of some selected NoSQL database based on the data model and some security features.

1. Redis

Redis is an open source database. In this type of database, key-value pairs are employed in storing data in the hash table for quick access. Due to its fast data processing applicability, it is used in high load cache access [11]. However, it doesn't to support data encryption i.e. communication is done in plain text, Also, access control is not employed instead a tiny level of authentication is offered. String escaping is not provided hence injection attacks are difficult [12] but possible under certain conditions [13].

2. Hbase

HBase is an open source database. It uses column oriented model adopted from Google big table and developed with Java, it depends on Apache Hadoop Framework [7] [14]. Both structured and unstructured data are supported by HBase.it employs distributed configuration and write-ahead logging which suits it more to an environment with regular write than reads. Authentication and authorization are provided by SASL (Simple Authentication and Security Layer) with Kerberos and ACL (Access Control List) respectively [15].

3. Mongo DB

Mongo DB is also an open source. It is document-oriented and implemented in C++.

It is designed for storing documents. Mongo DB employs an XML or JSON [14] format in managing documents, which makes it applicable in broad applications. It provides authentication and authorization as well as data encryption but susceptible to injection attacks.

4. Neo4J

Neo4j is an open source graph-oriented database. Data with features like a graph (connected) such as social networking, recommendation system are managed using this type of database. Nodes/vertices in the graph are shown by means of characteristics and the connections among nodes are typed and relationships can have their own relative properties. Neo4j is exceedingly versatile, it efficiently manages data with the extremely high relationship. It provides authorization using role-based access control (RBAC) as well as authentication. However, it doesn't support data encryption[16][17].

Table 1. Presents the security comparisons of some NoSQL systems.

Table 1. The security comparison of some open source nosql databases.

	Security features					
s/ no	***Database***	***Data model***	***Authenticaton***	***Authorization***	***Encryption***	***Input validation***
1	Redis	Key-value	Yes	Yes	No	No
2	Hbase	Column-oriented	Yes	Yes	No	No
3	Mongo DB	Document-oriented	Yes	Yes	Yes	No
4	Neo4j	Graph-oriented	Yes	Yes	No	No

3. Related Work

In this section, related works on security in NoSQL systems are discussed. Several types of research have been done to solve the security problems in cloud databases, however, contributions precisely for NoSQL databases are few. Below are some of the solution to security issues in NoSQL database.

In [18], a database security-as-a-service (DB-SECaaS) over a document-oriented database hosted in the cloud was proposed to ensure that access to the data is granted only to authorized users on a need-to-know basis. The system has a service-oriented architecture, which allows deploying different components for performing exact functions as distinct services. The DB-SECaaS provides authentication, fine-grained authorization, and encryption of the database objects. Authentication service is responsible of verifying users as well as requests by employing Strong authentication (SA), Identity management (IDM) and Certificate authority (CA). Fine-grained authorization service is responsible for protecting the data from unauthorized disclosure, it is provided through Policy administration point (PAP), Policy enforcement point (PEP) and Policy decision point (PDP) and finally, encryption service is responsible for performing the encryption and decryption of the data stored in the collection on the request of a privileged user, which is provided by advanced encryption standard (AES) along with Key distribution. The major security features of DB-SECaaS system achieves an effective and afford optimal protection to document-oriented NoSQL databases.

In [19], a Novel Security Extension for Redis NoSQL Database was proposed to provide security services. Redis in its simplest form is a key-value pair based data system. The extension provides authorization and authentication by employing the Rijndael method, Encryption services by using advanced encryption standard (AES), command restriction to set and get commands only, network security and persistent Data security using encryption. The system extends the capabilities of the Redis system to

complete multiuser scalable and secure database which can be used for real-time applications.

In [20], a secureNoSQL security scheme was proposed to secure queries over encrypted cloud NoSQL databases. The scheme employs a secure proxy to implement its security plan (designed in JavaScript Object Notation (JSON)). The security plan defines the procedure that will be used to maintain the security of data elements in a database and how to decode queries that are issued from particular user applications. It also specifies how those rules will be applied. These rules includes defining a collection(a group of NoSQL documents), choosing a crypto-system (Order Preserving Encryption (OPE) and the Advanced Encryption Standard(AES) modules) depending on the security policy of applications , the description of all sensitive data elements of database and specifies all cryptographic modules for all the sensitive data fields.

The SecureNoSQL achieves efficiency in terms of query processing time and preventing attacks.

In [21], a DIGLOSSIA tool was proposed to precisely and efficiently detect code injection attacks on server-side Web applications generating SQL and NoSQL queries. DIGLOSSIA is language independent, it employs a novel dual parsing method to compare the shadow string with the actual string generated by the application and ensure that the actual string does not contain any code tainted by user input. It operates in three stages, first stage creates a shadow character map and the dual parser, and secondly it computes a shadow value for each string that depends on user input, finally it detects injected code by inspecting and comparing the actual query string and its shadow based on two conditions: There is a one-to-one mapping between the parse tree of the actual query and the parse tree of the shadow query, specifically, all code in the actual query maps exactly to equivalent code in the shadow query and the shadow query does not contain any code in the original language L. The tool accurately detects SQL and NoSQL code injection attacks with virtually unnoticeable performance overhead.

In [1], a Nondeterministic finite automata (NFA) approach is proposed to detect and prevent NoSQL injection attacks on Mongo DB. The scheme operates in stages, in the first stage static analysis is employed to gather queries from the hotspot. In the second stage, the verification module dynamically compares user inputs during runtime with the corresponding input models. If the input is accepted the user is allowed to proceed else the user will be blocked from execution. The scheme was able to detect and prevent all attacks.

4. Security Testing Tool for Nosql Systems

The whole inputs into a NoSQL system can be broadly classified into two namely legitimate or illegitimate inputs. The legitimate inputs are those inputs accepted while illegitimate inputs are rejected because the inputs can be a form or medium used to attack the system. A Deterministic Finite Automaton (DFA) approach is employed to propose an input validation mechanism for a Mongo DB to detect and prevent NoSQL injection Attacks.

An automaton is an abstract self-propelled computing device which follows a scheduled sequence of operations automatically. An automaton with a finite number of states is called a Finite Automaton (FA) or Finite State Machine (FSM) [22].

Finite Automaton are categorized into two classes:

- Deterministic Finite Automaton (DFA)

• Non-deterministic Finite Automaton (NDFA / NFA)

A. Deterministic Finite Automaton (DFA)

In DFA, the transition from a state to another is determined for each input symbol. Therefore, it is called Deterministic Automaton. As it has a distinct symbol for the next number of states, the machine is called Deterministic Finite Machine or Deterministic Finite Automaton. For every symbol, a DFA has one possible transition. It also rejects any alphabet that fails to end in the final state [22].

B. Formal Definition of a DFA

A DFA can be represented by a 5-tuple $(Q, \sum, \delta, q_0, F)$ where:

- Q is a finite set of states.
- $\sum$ is a finite set of symbols called the alphabet.
- δ is the transition function where $\delta: Q \times \sum \rightarrow Q$
- q_0 is the initial state from where any input is processed ($q_0 \in Q$).
- F is a set of final state/states of Q ($F \subseteq Q$).

C. Graphical Representation of a DFA

A DFA is represented by digraphs called state diagram (Fig. 1):

The vertices represent the states.

- The arcs labeled with an input alphabet show the transitions.
- The initial state is denoted by an empty single incoming arc.
- The final state is indicated by double circles.

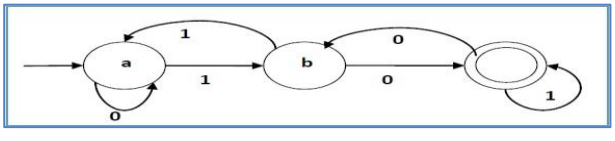

Fig. 1. Graphical representation of DFA [22].

Let a deterministic finite automaton be:

Q = {a, b, c},

- $\sum$ = {0, 1},
- q_0 = {a},
- F = {c}, and

Transition function δ as shown by the Table 2 below.

Table 2. Transition table [22].

Present State	Next State for Input 0	Next State for Input 1
A	a	b
B	c	a
C	b	c

D. Proposed Architecture.

An automata based (DFA) approach is used to develop a validation mechanism for a Mongo DB to detect and prevent NoSQL injection attack. The approach is similar to an implementation in [1] which employs NFA. DFA is chosen over NFA because it requires less time in execution and has been shown to have a performance advantage over NFA [23]. As earlier stated the DFA has one possible transition to a state for every symbol and also rejects any alphabet that fails to end in the final state. These two features are vital in our proposed architecture. The transitions from one state to another are mapped to input requests (i.e. alphabets in the DFA will represent our input request). A valid request will move through the states to the final state while the invalid request will fail to reach the final state. Hence, it will be rejected because it contains an invalid request which can be an attack on the system.

Figure 2 shows the proposed architecture of the input validation mechanism. It consists of two stages. In the first stage, valid prototypes of the request (expected type of query) are collected. In the second stage,

which is validation stage will test every input request from clients using the DFA approach against the expected types to check if it matches a valid request. A valid request will return the value 1, therefore it is accepted while an invalid request will return 0 hence it will be rejected.

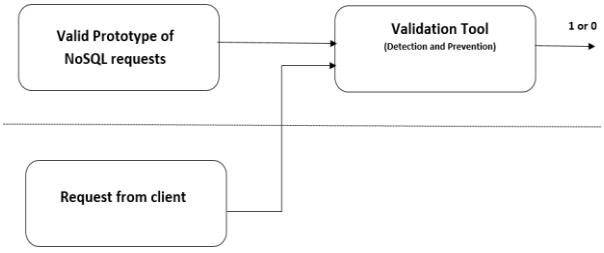

Fig. 2. Proposed Architecture of an Input validation tool for a NoSQL system.

In MongoDB, find(), insert(), remove() and update() are considered as the places from which a call goes to the database. An attacker or a malicious user will always try to make these calls to database to return true or 1 by manipulating the inputs. As an example to show how the proposed mechanism work, supposing a Mongo DB database is used to store details about books such as title of the book, authors, ISBN numbers, publishers, etc. A user searches for a book by entering ISBN numbers in input box and if the number is correct, the system will display the information of the book. An example of valid request will be:

db.collection.find ({ISBN:isbn_number});

Injection attacks are mostly performed through an input field or the URL.For our example we shall use the first way i.e the input field, let's assume a system is developed with PHP and JavaScript. Passing value and query can be done with the code below:

$unsearch= $_POST ['name'];

$jss= " function () {return this.ISBN == '$unsearch';}";

$cursor= $user -> find (array ('$ where => $jss));

The system accepts the value from the input field by $_POST['name'] and then forward it to javascript $jss. Then the system begins to query the system by using the ISBN number: return this.ISBN. At this instant, the system commence searching the database to match the ISBN number by executing find (array ('$where'=> $jss)).If the argument in find() is true then it will display the book details.

A malicious user could inject a bit of code to make the query always true when searching document by document. Let's assume the ISBN number is 587952, the attacker can append the OR operator to the ISBN: i.e 587952 ′ ‖ ′1′ ==′1. After inserting ′ ‖ ′1′ ==′1, whatever the attacker input it will always return true.

The proposed validation mechanism will detect the additional code because it does expect it. The DFA expects only the query operator and number. Below is a graphical illustration of our DFA model (Fig. 3).

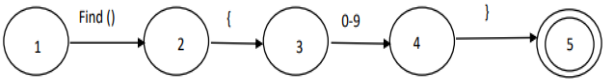

Fig.3. Sample model for MongoDB query find ({587952}).

For any input other than the expected operator or the ISBN number, the model will not end in the final state. Hence the request is rejected.

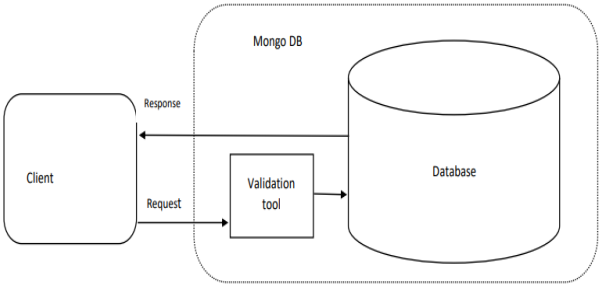

Fig. 4. Simple architecture of the deployed validation tool in Mongo DB.

Figure 4 shows a simple architecture of the deployed validation tool in Mongo DB. The security testing tool can be developed as a component. It will improve the security of NoSQL system because invalid inputs requests will be detected and prevented from being processed.

5. Conclusion

Improving the security of NoSQL system is imperative due to their growing importance. In this paper, an architecture for a security test tool has been proposed. It utilizes the features of a Deterministic Finite Automaton (DFA) to detect and prevent NoSQL injection attacks by validating input request through comparing it with a valid prototype of the expected request. A valid request is allowed to be processed while an invalid request will be rejected to protect the system from attacks. However, the model is a proof of concept and is limited to only the find () operator in Mongo DB.

In addition, a comparison based on some security features of NoSQL systems such as authentication, authorization, data encryption and input validation are provided based on recent literature. The proposed architecture will no doubt enhance the security of Mongo DB systems.

As a future work, an implementation of both DFA and NFA based systems will done and evaluations will be carried out.

Acknowledgment

The authors will like to thank the faculty of computing and information technology (FCIT), King Abdulaziz University Jeddah, KSA for its continuous support.

References

[1] **Joseph, S.** and **Jevitha, K. P.**, "An Automata-Based Approach for the Prevention of NoSQL Injections," in *Security in Computing and Communications. SSCC 2015.*, 2015, vol. vol 536, pp. 538–546.

[2] **Okman, L., Gal-Oz, Gonen, N. Y., Gudes, E.** and **Abramov**, **J.**, "Security issues in NoSQL databases," in *Proc. 10th IEEE Int. Conf. on Trust, Security and Privacy in Computing and Communications, TrustCom 2011, 8th IEEE Int. Conf. on Embedded Software and Systems, ICESS 2011, 6th Int. Conf. on FCST 2011*, 2011, pp. 541–547.

[3] **Aviv Ron, A. P.** and **Shulman-Peleg**, **Alexandra,** "Analysis and Mitigation of NoSQL Injections," *IEEE Secur. Priv.*, vol. 14, no. 2, pp. 30–39, 2016.

[4] **Neo4j**,[Online].Available: https://www.owasp.org/index.php/Data_Validation.

[5] **OWASP**,[Online]. Available: https://neo4j.com/docs/operations-manual/current/security/authentication-authorization/introduction/.

[6] **Lith, M. J. A**, "*Investigating storage solutions for large data: A comparison of well performing and scalable data storage solutions for real time extraction and batch insertion of data*.," Chalmers University of Technology, 2013.

[7] **Chang, G. R. F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T.** and **Fikes A**, "Bigtable: A distributed storage system for structured data.," *ACM Trans Comput Syst*, vol. 26, no. 2, 2008.

[8] **DeCandia, V. W. G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A.** and **Pilchin, A.**, Sivasubramanian S, Vosshall P, "Dynamo: amazon's highly available key-value store.," in *ACM SIGOPS Operating Systems Review.*, 2007, p. pp 205–220.

[9] **Cattell, R.,** "Scalable SQL and NoSQL data stores.," *ACM SIGMOD Rec.*, vol. 39, no. 4, pp. 12–27, 2011.

[10] **Du, J., Han, J. Haihong, E.** and **Le, G.** "Survey on NoSQL database," in *6th International Conference On Pervasive Computing and Applications (ICPCA)*, 2011, pp. 363–366.

[11] **Noiumkar, P.** and **Chomsiri**, **T.,** "A Comparison the Level of Security on Top 5 Open Source NoSQL Databases," in *9th International Conference on Information Technology and Applications(ICITA2014)*, 2014.

[12] "**Redis**." [Online]. Available: https://redis.io/topics/security.

[13] **Spiegel**, **Patrick,** "*nosql injection redis*." [Online]. Available: https://medium.com/@PatrickSpiegel/https-medium-com-patrickspiegel-25b332d09e58.

[14] **P. A. Kuznetsov S.,** "Nosql data management systems," *Progr. Comput Softw*, vol. 40, no. 6, pp. 323–332, 2014.

[15] **HCC**, "*Hbase security model*." [Online]. Available: https://community.hortonworks.com/articles/72980/hbase-security-model.html.

[16] Neo4j, [Online]. Available: http://neo4j.com.

[17] **Grolinger, M. A. C. K., Higashino, W.A.** and **Tiwari, A.,** "Data management in cloud environments: NoSQL and NewSQL data stores," *J. Cloud Comput. Adv. Syst. Appl.*, 2013.

[18] **Ghazi, Y., Masood, R., Rauf, A., Shibli, M. A.** and **Hassan**, **O.,** "DB-SECaaS: a cloud-based protection

system for document-oriented NoSQL databases," *EURASIP J. Inf. Secur.*, vol. 2016, no. 1, p. 16, 2016.

[19] **Zaki, A. K.** and **Indiramma**, **M.,** "A novel redis security extension for NoSQL database using authentication and encryption," *Proc. 2015 IEEE Int. Conf. Electr. Comput. Commun. Technol. ICECCT 2015*, 2015.

[20] **Ahmadian**, **M.,** "Secure query processing in cloud NoSQL," in *2017 IEEE International Conference on Consumer Electronics, ICCE 2017*, 2017, pp. 90–93.

[21] **Son, S., McKinley, K.** and **Shmatikov**, **V.,** "Diglossia: detecting code injection attacks with precision and efficiency," *Proc. 2013 ACM ...*, no. 2, pp. 1181–1191, 2013.

[22] **Point**, **Tutorial,** "Automata Theory Tutorial." [Online]. Available: https://www.tutorialspoint.com/automata_theory/deterministic_finite_automaton.htm.

[23] **Tsotras, V. J., Moro, M. M.** and **Z. Vagena**, "*Twig query processing over graph-structured XML data*," pp. 43–48, 2004.

أداة اختبار الأمان لأنظمة NoSQL

محمد أمين لول و **مصطفى السيد صالح**

قسم نظم المعلومات، كلية الحاسبات وتقنية المعلومات، جامعة الملك عبدالعزيز، جدة، المملكة العربية السعودية

mlawal@stu.kau.edu.sa

المستخلص. أصبحت أنظمة NoSQL أكثر شيوعًا بسبب مزاياها المتأصلة وحلولها التي توفرها لحدود قاعدة البيانات العلائقية. ومع ذلك، على الرغم من فوائدها، فإنها تأتي مع التحديات الأمنية. في هذه الورقة، تم اقتراح بنية لآلية التحقق من صحة الإدخال لـ Mongo DB للكشف عن هجمات حقن NoSQL ومنعها، وتستخدم الآلية منهجية (Deterministic Finite Automaton DFA) لاكتشاف ومنع الهجمات على أنظمة NoSQL. مزيدًا على ذلك، يتم توفير مقارنة الأمان بعض أنظمة NoSQL استنادًا إلى الأدبيات الحديثة. وميزات الأمان التي تمت مقارنتها هي المصادقة والترخيص وتشفير البيانات والتحقق من صحة الإدخال. ستعمل الآلية المقترحة على تحسين أمان نظام Mongo DB لأنه سيتم اكتشاف طلبات غير صحيحة ومنع معالجتها أو تنفيذها.

الكلمات المفتاحية: NoSQL، Mongo DB، DFA، اختبار الأمان.