Comparative Security Vulnerability Analysis of NoSQL and SQL Database Using MongoDB and MariaDB

Jeremiah Oluwagbemi Abimbola#1, Osuolale A. Festus#2

Department of Computer Science, Changchun University of Science and Technology, Changchun
Department of Computer Science, Federal University of Technology, Akure

Abstract
Security is a growing concern for any system from desktop to web applications. As data increases, there must be a database system that can safely deal with the current need. However, with update in database systems, comes vulnerabilities. Hence, the need to continuously keep this research in motion. This paper presents a comprehensive study of potential security vulnerabilities and challenges for two databases. A detailed comparison between traditional SQL and NoSQL databases is provided and identification of a set of vulnerabilities specific to representative database applications using MongoDB and MariaDB.Examples of attacks and mitigation techniques are also provided, the discussion and results shown helps database administrators and application developers increase awareness of arising threats while deploying SQL and NoSQL databases.

Keywords: *security, database, SQL, NoSQL, vulnerability*

I. INTRODUCTION

Data is a typical representation of relevant aspects of reality (for example, a student record), in a way that helps processes requiring this information (for example, finding a student record in a school management system).The model of data is such that it is organized into rows, columns and tables, and indexed to make it effortless to find relevant information. Data gets updated, incremented and deleted as new information is added [1].Databases are popularwithlargesystems and critical systems, such as aviation systems but are also present in smaller distributed workstations and midrange systems, such aseducation systems, personal computers etc.Hence, the importance of a high level of security.Data security is an important feature for any information system. [2]

Structured Query language (SQL) **pronounced as "S-Q-L" or sometimes as "See-Quel**"isthe standard language for dealing with Relational Databases.SQL programming can be effectively used to insert, search, update, delete database records and more. On the other hand, NoSQL means "Not only SQL", it is an upcoming category of Database Management Systems. Its main characteristic is its non-adherence to Relational Database Concepts. NoSQL database are non-relational databases that scale out better than relational databases and are designed with web applications in mind.They do not use SQL to query the data and do not follow strict schemas like relational models as seen in Figure 1 below.With NoSQL, ACID (Atomicity, Consistency, Isolation, Durability) features are not assured always.SQL databases have been in use since the 1970s when it was first developed but recently, there's been more advocacy for the NoSQL database because of its high performance, and ability to handle complex task in real time. Consider this example, Imagine that you have coupons that you wanted to push to mobile customers that purchase a specific item. This customer facing system of engagement requires location data, purchase data, wallet data, and so on. You want to engage the mobile customer in real-time.What you require is a very agile delivery system that is easily able to processes unstructured data. Although, with simplicity of data representation and ability to handle real-time data comes security shortcomings.

Previous research carried out on security vulnerability analysis of databaseare limited to the release at the time the research was conducted. [3]-[5]but as more updates are released, more deficiencies are introduced, therefore there is a strong need for continuous security vulnerability analysis.

The aimof this research is to examine in depth security arising concerns from recentSQL and NoSQL databases that may be deployed by web applications and other critical systems. Security concern areas include encryption, internodecommunications, authentication, authorization, audit and data consistency. In this project, we will consider the MongoDB for the NoSQL database and MariaDB for the SQL (relational) database. MongoDB stores data in flexible, (JavaScript Object Notation) JSON-like documents, meaning fields can vary from document to document and data structure can be changed over time. The document model maps to the objects in your application code, making data easy to work with. Ad hoc queries, indexing, and real time aggregation provide powerful ways to access and analyze your data. It is a distributed database at its

core, so high availability, horizontal scaling, and geographic distribution are built in and easy to use.MariaDB on the other hand, is an open source relational database management system (DBMS) that is a compatible drop-in replacement for the widely used MySQL database technology. It was created as a software fork of MySQL by developers who played key roles in building the original database.We will look at important areas of security lapsesof the both databases and the possible threats that could arise. We will also recommend possible defense mechanisms for security attacks by usingsome code snippets.

This paper is organized as follows: Section 2provides an overview ofNoSQL and SQL securityissues. Sections 3and 4 discuss somecommon security issues present in MongoDB,and MariaDB databasesrespectively.Section 5 provides some examples of attacks and possible solutions.Finally, Section 6 concludes the paper.

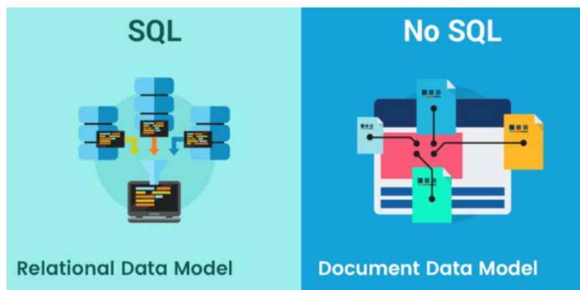

Figure 1.0 Differences between SQL and NoSQL
Source:Apptunix

II. OVERVIEW OF NOSQL AND SQL SECURITY ISSUES

NoSQL databases become very prone to exploits once attackers are able to identify software vulnerability. Incomplete input validation, weak and exposed authentication, errors in the application level permissions handling, insecure communication, unauthorized access to unencrypted data, etc. are some of the vulnerabilities applicable for NoSQL. The same applies to the SQL databases, as most issues occur minutes after deployment [3], ranging from stolen backups to SQL injection issues.
Generally, insecure connection between web application and database, insufficient support for specialauthorized users (e.g., DBA) and insufficient authentication are known vulnerabilities and truthfully, there are no definite or standard principles for achieving best practices for authentication, authorization and encryption. [4] Some of the issues in NoSQL and SQL can be summarized as follows. We discuss them separately forMongoDB and MariaDB in this section.

- Encryption
- Inter node communications
- Authentication
- Authorization
- Audit

III. SECURITY ISSUES IN MONGODB ENCRYPTION

One of the most serious problems of MongoDB is that data files are not natively encrypted over the wire by default. Although, recent development of mongoDB has added a layer of encryption using Atlas but it is only available at rest. If encryption is required for the data file, theapplication layer needs to encrypt the data beforesending it to the database server and this process is often very difficult and costly. [5]-[6]

Data-in-Motion (client-nodecommunication)

Recently, MongoDB by default now support TSL (Transport Layer Security) and SSL (Secure Socket Layer) client-node communication. To use SSL, it is required to recompile whole MongoDB with the “-sl" option or deploy MongoDB enterprise version. However, before the data is sent and after the data arrives at its endpoint, the data appears unencrypted, or “in the clear”[5]. This is a great gap in that there could be some passive attack in the background at the point data is unencrypted.Further steps to generate keys are needed for configuring client/server for TSL and SSL communication.

Authentication

By default the DB installs with NO password credentials! Reading the MongoDB manual, the MongoDB developers have to entirely take into consideration security. It then lies in the hands of the application developers and running it in a trusted environment.Authentication is disabled by default. This MongoDB on its own does provide support for authentication on a per-database level. Users exist in the context of a single logical database. Latest version ofMongoDB 3.0 with Atlas implementation has seen an inclusion of Authentication support with LDAP. It does not support the authentication if it runs in shared mode. Thus, for both standalone mode and replica-set mode, the authentication should be activated on MongoDB in order to authenticate each server before joining the cluster [4]

Authorization

Authorization is disabled by default in MongoDB. This implies that any created user has read-only access to the entire database. That essentially means that once you have a user, you have provided access by default to everything stored in the database. It provides authorization on a per-database level by using a role-based access control approach. Available roles are limited to the following: read, readWrite, readAnyDatabase,readWriteAnyDatabase, userAdmin, clusterAdmin,userAdminAnyDatabase, dbAdmin, and dbaAdminAnyDatabase.Also, a user

with access to the Admin database has read/write access to just everything. There is no granularity, and by default there is no Admin password therefore by default we have access to everything. [7] This is a serious security issue.

Audit

Latest MongoDB documentation states that there is an adequate auditing facility implementedfor security logging and monitoring, authentication, authorization and CRUD operations [5] but most monitoring and reporting tools currently distributed with MongoDB are related to database performance forshowing the current state of a MongoDB instance.There is an HTTPConsole for each MongoDBinstance to show information about the system andconnecting clients. However, if security is notenabled for the MongoDB instance, no authorizationis needed to access this interface, resulting in alikelyexposure of audit data. Also if the server terminates before it commits the event to the audit log, the DB may lose the event thereby making any security flaw untraceable. [8]

IV. SECURITY ISSUES IN MARIADB ENCRYPTION

MySQL does not support database encryption. However, the encryption can be done at applications level and has provided many inbuilt methods. This is a challenge for MariaDB because it is a fork of MySQL. Although, the enterprise version ofMariaDB provides data-at-rest encryption. Theencryption provides no additional protection against threats caused by authorized database users. Specifically, SQL injections aren't prevented[9].

Data-in-Motion (Inter-node communication)

With the advancement of cluster computing, MariaDB was updated in its later releases to provide shared data storage environment for clusters.By default, Galera Cluster replicates data between each node without encrypting it [10]. To mitigate this concern, the data can be transferred between nodes using Transport layer Security (TSL).

Authentication

The authentication of users is delegated to plugins.Two plugins are always available: mysql_native_password and mysql_old_password - they implement the compatible MySQL password authentication with 20 byte (Used in MySQL 4.0 or later) and 9 byte (used in MySQL 3.23) scrambles. In Secure Password Authentication, MYSQL uses SHA1 hash function for storing passwords of users.

Authorization

Authorization is provided by default on MariaDB, i.e. the database user doesn't have right to all the privileges until is enabled. To do this, the user must be logged in which provides a layer of security.

Auditing

There is an audit plugin built into MariaDB, it provides an easy to use, policy based auditing solution that helps organizations implement greater security mesaures and satisfy regulatory compliance and every action is logged. This includes authorization access and authentication processes such as database access etc.

V. ATTACK EXAMPLES

A. Attacks on MongoDB and Possible Defense Mechanisms

Injection Attack: The MongoDB API expects BSON (Binary JSON) calls, and includes a secure BSON query assembly tool. However, according to MongoDB documentation, un-serialized JSON and JavaScript expressions are permitted in several alternative query parameters. To prevent attacks, web developers must apply proper filtration/validation on all forms.

Tautologies. These attacks allow bypassing authentication or access mechanisms by injecting code in conditional statements, thereby generating expressions that are always true (*tautologies*). Attackers can exploit this to log in to the system without appropriate credentials.

For example, in a login scenario, using php and mongoDB, since php has a built-in mechanism for associative arrays that lets attackers send the following malicious payload:

```
db->logins-
>find(array("username"=>$_POST["
username"],
"password"=>$_POST["password"]))
;
```

the attacker could do this

```
username[$ne]=1&password[$ne]=1
array("username" => array("$[ne]
" =>1), "password" =>array("$ne"
=> 1));,
```

Because $ne is MongoDB's not equals condition, it queries all entries in the logins collection for which the username is not equal to 1 and the password is not equal to 1. Thus, this query will return all users in the logins collection. To mitigate this issue, we need to cast the parameters received from the request to the proper type, in this case, using the string

```
db->logins->find(
array("username"=>(string)$_POST
["username"],
"password"=>(string)$_POST["pass
word"]));
```

JavaScript injections. Recently, there's been more power given to JavaScript such that it doesn't only function for client data manipulation but now for server. This new type of vulnerabilities found inMongoDB allows execution of JavaScript in the

database context. JavaScript enables complicated transactions and queries on the database engine. [11]Passing unsanitized/unresolved user input to these queries might allow for injection of arbitrary JavaScript code, which could result in illegal data extraction or alteration. Consider this instance, if a function is written like this (MapReduce)that takes the field name that it should act on (amount or price) as a parameter from the user. In PHP, such code can look like this (where $param is user input): because user input isn't being escaped here, a malicious input (that might include arbitrary JavaScript) will execute.

```
$map = “function() {
    for (var i = 0;
i<this.items.length; i++) {
        emit(this.name,
this.items[i].$param);
    }
}”;
$reduce = “ function(name, sum)
{
    return Array.sum(sum);
}”;
$opt = “ {
    out: ‘totals’
}”;
$db - >
execute(“db.stores.mapReduce($ma
p, $reduce, $opt);”);
```

The proper solution to such an attack is to disable the use of JavaScript execution in the database configuration. If JavaScript must be used, it's best practice not to use any user input in its formation.

Origin violation. HTTP REST APIs are a popular module in NoSQL databases;such as it is in Mongo however, they instigate a new type of exposure that permits attacks on database from another domain. [11] In cross-origin attacks, attackers exploit legitimate users and their Web browsers to perform an unwanted action.

DOS Attack: (Denial of Service) attacks have high possibilities in MongoDB. By default MongoDB does not require authentication and authorization, therefore an attacker can use valid user credentials and he/she does not have to be an administrator to carry out the attack.

B. Attacks on MariaDB and Possible Defense Mechanisms

SQL Injection Attack: One of the most common attacks on MariaDB is SQL injection attack. Attacks starts from simply accepting user input. Opportunities for SQL injection typically occur on users entering data like a username, and the code logic failing to evaluate this input. The Code, instead, allows an attacker to insert a MariaDB statement, which will run on the database.A good way to combat this attack could be to use regular expressions to perform validation through pattern matching. For instance, the code below uses regular expression to check for just the right username and nothing else [12].

```
if  (check_match("/^\w{8,20}$/",
$_GET['username'], $matches)) {
    $result                    =
mysql_query("SELECT  *FROM  users
WHERE username = $matches[0]");
} else {
    echo "Invalid username";
}
```

VI. CONCLUSION

In many critical systems today including web applications, database plays a very vital role in safely keeping and retrieving information. In this paper, a summary of two databases has been given based on security vulnerabilities and strengths. As new database systems are developed, it is also crucial to evaluate them based on the criteria listed in this paper. Each database has its drawbacks and advantages as well and the choice of any database depends on the robustness of the system. A summary of the analysis is provided below.

TABLE 1
COMPARISON BETWEEN MONGODB AND MARIADB

Criteria	**Mongo DB**	**Maria DB**
Encryption	Data files are not natively encrypted over the wire by default.	The encryption provides no additional protection against threats caused by authorized database users. Specifically, SQL injections aren’t prevented
Inter-node communication	Before the data is sent and after the data arrives at its endpoint, the data appears unencrypted	Updated MariaDB provides shared data storage environment for clusters. By default, Galera Cluster replicates data between each node without encrypting it

Authentication	Authentication is disabled by default, the DB installs with NO password credentials	The authentication of users is delegated to plugins. Once these plugins are vulnerable, the data also becomes vulnerable
Authorization	Authorization is disabled by default in MongoDB. This implies that any created user has read-only access to the entire database.	Authorization is provided by default on MariaDB
Audit	If security is not enabled for the MongoDB instance, no authorization is needed to access the audit interface, resulting in a likely exposure of audit data	There is an easy to use audit plugin built into MariaDB which provides, policy based auditing solution that helps organizations implement greater security measures.

The overall study discovers that the configuration of any database determines how secured it is, because even a good database can be wrongly configured, thereby causing loop-holes. Some defense mechanisms have also been mentioned to guide database administrators or software developers who use database to ensure secured data at all times. Finally, the need to follow standards and best practices when using any database either SQL or NOSQL cannot be over-emphasized and the use of stronger encryption algorithm to encrypt files (NoSQL) even at rest.

REFERENCES

[1] M. Rouse, "Search SQL Server," 19 February 2019. [Online]. Available: https://searchsqlserver.techtarget.com/definition/database. [Accessed 15 April 2019].

[2] X. Wei, "Analysis of Web-based Network Database Security Technology," Agricultural Technology and Equipment, vol. 02, pp. 32-34 , 2019.

[3] C. Osborne, "ZDNet," 23 June 2013. [Online]. Available: https://www.zdnet.com/article/the-top-ten-most-common-database-security-vulnerabilities/. [Accessed 1 May 2019].

[4] Hossain Shahriar, Hisham M. Haddad, "Security Vulnerabilities of NoSQL and SQL Databases for MOOC Applications," International Journal of Digital Society (IJDS), vol. 8, no. 1, March 2017.

[5] MongoDB, "MongoDB," [Online]. Available: https://docs.mongodb.com/manual/core/security-encryption-at-rest/. [Accessed 1 May 2019].

[6] TownsendSecurity, "Townsend Security," [Online]. Available: https://info.townsendsecurity.com/mongodb-encryption-key-management-definitive-guide. [Accessed 1 May 2019].

[7] David Kirkpatrick, "Mongodb - Security Weaknesses in a typical NoSQL database," 21 March 2013. [Online]. Available: https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/mongodb-security-weaknesses-in-a-typical-nosql-database/. [Accessed 1 May 2019].

[8] MongoDB, [Online]. Available: https://docs.mongodb.com/manual/core/auditing/. [Accessed 1 May 2019].

[9] Maria Db, "Why Encrypt MariaDB Data?," [Online]. Available: https://mariadb.com/kb/en/library/why-encrypt-mariadb-data/. [Accessed 9 May 2019].

[10] "Securing Communications in Galera Cluster," [Online]. Available: https://mariadb.com/kb/en/library/securing-communications-in-galera-cluster/. [Accessed 9 May 2019].

[11] Aviv Ron, Alexandra Shulman-Peleg ,Anton Puzanov , "Analysis and Mitigation of NoSQL Injections," 18 Jan 2017. [Online]. Available: https://www.infoq.com/articles/nosql-injections-analysis. [Accessed 12 May 2019].

[12] Tutorial Point, "MariaDB - SQL Injection Protection," [Online]. Available: https://www.tutorialspoint.com/mariadb/mariadb_sql_injection_protection.htm. [Accessed 12 May 2019].

[13] G. Menegaz, "Zdnet," 1 October 2012. [Online]. Available: https://www.zdnet.com/article/what-is-nosql-and-why-do-you-need-it/.

[14] A. W. M.W. Grim, "Security and Performance Analysis of Encrypted NoSQL Databases," Security of Systems and Networks, pp. 10-14, 12 February 2017.

[15] O. H. A.-T. H. M. E.-B. A. S. S. Ahmed M. Eassa, "NoSQL Racket: A Testing Tool for Detecting NoSQL Injection Attacks in Web Applications," (IJACSA) International Journal of Advanced Computer Science and Applications, vol. 8, no. 11, p. 615, 2017.

[16] Z. Xiangrong, "Analysis of Database SQL Injection and Its Security Protection," Journal of Taiyuan University (Natural Science Edition), vol. 35, no. 03, pp. 60-62+76, 2017.

[17] Y. Xiaoyan and G. Mei, "Research on NoSQL Non-relational Database Security Based on Hadoop," Microcomputer applications , vol. 34 , no. 12, pp. 43-45, 2018.

[18] Apptunix, https://www.apptunix.com/blog/sql-or-nosql-database/.