Security Vulnerabilities of NoSQL and SQL Databases for MOOC Applications

Hossain Shahriar[1], Hisham M. Haddad[2]
[1]*Department of Information Technology*
[2]*Department of Computer Science*
Kennesaw State University, USA

Abstract

Massive Open Online Courses (MOOCs) are popular among learners for free or low cost access to education materials. It is estimated that millions are currently users of various courses offered through MOOCs. These courses are deployed on open source technologies. With the increasing data generated while interacting with users, many of the deployed platforms have already switched to NoSQL database platform. However, there is little discussion on the risks and threats that NoSQL related technologies pose to learners. This paper presents an in depth study of potential security vulnerabilities and challenges for databases. We provide detailed comparison between traditional SQL and NoSQL databases and identify a set of vulnerabilities inherent to representative database applications like MongoDB and Cassandra. We also provide examples of attacks. The discussion helps learners and application developers to increase the awareness of threats arise while interacting online with platforms deploying NoSQL databases.

1. Introduction

Massive Open Online Courses (MOOCs) are popular among learners seeking low cost access to learning materials. Some MOOC providers such as Coursera [11], EdX [12] and Udacity [13] have gained popularity. An estimated of over 7 million students in the United States alone have taken a minimum of one online course [14].

MOOCs are special web applications intended for learning management system. Earlier, we have conducted studies on learning analytics support and common security concerns for cloud applications and privacy [20, 21]. This paper examines in depth security arising concerns from databases that may be deployed by MOOC web applications.

One common goal of having databases is to store and retrieve data. The data can be stored in relational databases (SQL) like Oracle, DB2, SQL server, and MySQL. However, the huge amount of data could be also stored in non-relational database (NoSQL). It is well known that as the size of data and files become larger, traditional SQL database do not work very well [1]. The limitless array of data collection technologies moved from simple online actions to point of sale systems to GPS tools to smartphones and tablets to sophisticated sensors. To perform operations on big data, the database needs to be more scalable and robust.

Currently, there are more than dozens of NoSQL databases. However, Cassandra [22] and MongoDB [16] are two popular choice of NoSQL database used by MOOC applications.

The security concerns remain prevalent for MOOC learners. The stored data in MOOC application's databases raise the issue of confidentiality and privacy of the data. In this paper, we present a survey of common security concerns for both relational and non-relational databases. We also provide some guidelines to mitigate them. We review vulnerabilities in two common NoSQL databases used with MOOC applications (Cassandra and MongoDB) based on the literature [6-10, 17, 18]. We also discuss vulnerabilities in MySQL.

This paper is organized as follows: Section 2 compares SQL and NoSQL. Section 3 provides an overview of MongoDB and Cassandra. Section 4 provides an overview of NoSQL security considerations. Sections 5, 6, 7 discuss some common security issues present in MongoDB, Cassandra, and MySQL databases, respectively. Section 8 provides some examples of attacks. Finally, Section 9 concludes the paper.

2. Comparison between SQL and NoSQL

To understand NoSQL, we first identify some differences between SQL and NOSQL. SQL works on structured data to store data in certain tabular format and have relations with other tables to perform join operations. NoSQL is suitable for unstructured data [2,3,4]. It stores data in unstructured format and very flexible to perform operations in cluster computing environments. Below we discuss the differences.

a) Data model: SQL databases stores data in tables with rows and columns. Tables can be related to one another and cooperate during data storage and retrieval. On the other hand, NoSQL databases are not intended for storing data in rows and columns of tables. However, they store data in different formats and grouped together in chunks. Non-relational data is often stored as collections (e.g., documents) which support JSON (JavaScript Object Notation), key-value pairs or graphs. Figure

1 shows a high-level difference between SQL and NoSQL data model.

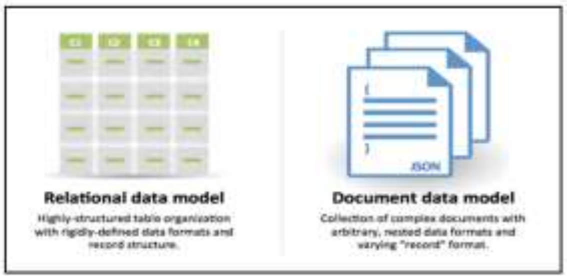
Relational data model
Highly-structured table organization with rigidly-defined data formats and record structure.
JSON
Document data model
Collection of complex documents with arbitrary, nested data formats and varying "record" format.

Figure 1. Data model in MySQL and NOSQL

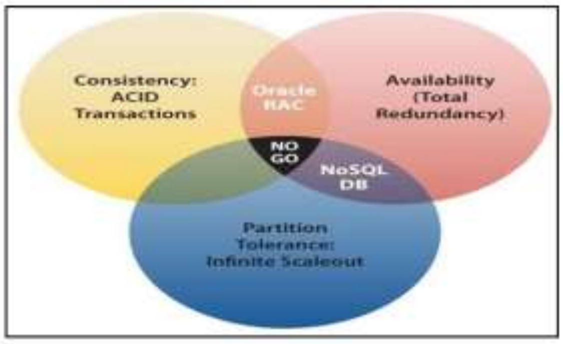
Consistency: ACID Transactions
Oracle RAC
Availability (Total Redundancy)
NO GO
NoSQL DB
Partition Tolerance: Infinite Scaleout

Figure 2. CAP Theorem

b) Schema: The tables have a pre-defined schema (column definitions) to describe the data in SQL. This gives data modeling paramount importance, and necessitates the 'Get it right first time' attitude. Though pre-defined schema is stable and reliable, changing the schema on an existing data in the table is very difficult. Non-relational data, on the other hand, thrives on dynamic schemas and is referred to as unstructured data. Non-relational data can easily accommodate changes in data type/structure due to its dynamic schema support.

c) Normalization: Data stored in relational databases aim to provide high normalization – dividing data into smallest logical tables to prevent duplication and to retrieve data using joins. This involves high computations and adds complexity. NoSQL databases, on the other hand, are stored in flat collections, where data might often be duplicated. A single chunk of data is seldom partitioned off, rather stored as an entity, thus allowing easier reads/writes to that single entity.

d) Scalability: The main difference between SQL and NoSQL is the degree to scale the infrastructure. SQL databases support vertical scalability where it is possible to only add more processors and memories within the system. On the other hand, NoSQL supports horizontal scaling. Power is improved by adding more nodes into clusters. NoSQL databases can scale according to the web-scale processing needs.

e) Data Manipulation: Relational databases manipulate data using SQL queries. They perform CURD operations for data manipulation. NoSQL databases use Unstructured Query Language (UnQL) to perform CURD operations. NoSQL uses limited and effective access patterns.

f) Integrity: SQL databases maintain integrity through four well known properties: Atomicity, Consistency, Isolation, and Durability (ACID). However, NoSQL databases are capable of providing Consistency, Availability, Partition Tolerance (CAP [5]) properties. Figure 2 shows the overlap between ACID and CAP.

SQL databases can store and handle data reliably. The biggest proposition for NoSQL aims to handle problems related to big data. As the data is not structured and there are no operations like join and normalization, the work load is low. The features that are available in NoSQL enable to meet big data requirements in the present scenario. Table 1 provides an overview of the differences between SQL and NoSQL.

Table 1. Differences between SQL and NoSQL

Characteristic	**SQL**	**NoSQL**
Data Storage	Stores information in the form of tables. Each row contains information about one specific entity. Columns store separate data items.	Data is stored in different formats in various databases that include key-value pairs, documents, graphs, etc.
Schemas	Creation of table is based on schemas and it is complex task to alter the schema once defined.	Schema is highly dynamic and information can be changed easily. It is flexible compared to relational databases.
Scalability	Scaling is vertical. It is possible to scale a RDBMS across multiple servers, but it is difficult and time-consuming process.	NoSQL is horizontally scalable. More servers can be added to increase the performance.
Integrity Compliance	The vast majority of relational databases are ACID compliant.	NoSQL databases sacrifice ACID compliancy for performance and scalability.

3. Overview of MongoDB and Cassandra

3.1. MongoDB

MongoDB has rapidly grown to become a popular database for web applications and clients, backend and database layer. The database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.

MongoDB is a collection of documents. This is equivalent to a RDBMS table. The collection is present within a database. A collection of documents can contain a variety of fields. Each document may have different fields.

An example of MongoDB query for inserting data is shown below. Here, each `column:value` pair defines the property of a document.

```
db.mycol.insert({
   _id: ObjectId(7df78ad8902c),
   title: 'MongoDB Overview',
   description: 'MongoDB is no sql
database',
   by: 'Paper',
   tags:
['mongodb','database','NoSQL'],
})
```

3.2. Cassandra

Apache Cassandra is a top level Apache project born at Facebook and built on Amazon's Dynamo and Google's BigTable, which is a distributed storage system for managing very large amounts of structured data spread out across many commodity servers, while providing highly available service with no single point of failure.

A column family is a container for an ordered collection of rows. Each row, in turn, is an ordered collection of columns. Data is stored in the form of nested key value pair. Column family is equivalent to a table in RDBMS. All the queries are the same as SQL and are represented as tables. Cassandra Query Language (CQL) is used to perform database operations. An example of insertion statement of CQL is shown below.

```
INSERT INTO data(name, email) VALUES
('john',['jdoe@test.com','jdoe2@test.c
om'])
```

4. NoSQL Security Consideration

NoSQL databases may become susceptible to exploits once attackers are able to identify security or software weaknesses. Insufficient or ineffective input validation, errors in the application level permissions handling, weak authentication, insecure communication, illegal access to unencrypted data, etc. are some of the vulnerabilities applicable for NoSQL.

In general, insecure connection between web application and database, insufficient support for special authorized users (e.g., DBA) and insufficient authentication are known vulnerabilities. There are no standard principles for best practices for authentication, authorization and encryption. The current practices in the field usually involve placing the security in the middleware layer, and ignoring security on the cluster level.

Some of the issues in NoSQL can be summarized as follows. We discuss them separately for MongoDB and Cassandra in next two sections.

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

Below we show one specific code injection example for MongoDB.

4.1. NoSQL Injection

Similar to the traditional RDMBS counterparts, NoSQL databases are susceptible for query injection attacks; especially those heavily use server-side JavaScript and PHP to enhance database performance. Let us consider MongoDB. Its internal operator "$where", acts as a filter like the "where" clause in a SQL query. It can accept sophisticated JavaScript functions to filter data. An attacker can pass arbitrary code or commands into the $where operator as part of the query.

Other vulnerable MongoDB operations include db.eval(), map Reduce, and group, which permit running arbitrary JavaScript expressions on the server. NoSQL injection attacks will execute where the attack string is parsed, evaluated, or concatenated into NoSQL API calls. Attackers need to be familiar with the syntax, data model, and underlying programming language of the target database in order to design specific exploits.

The following examples demonstrate how JavaScript NoSQL injections can be crafted against a vulnerable MongoDB instance. Consider the following two valid, equivalent JavaScript statements to retrieve a collection of data that meet the condition (`credits < debits`).

```
1. db.myCollection.find ({ $where:
   "this.credits < this.debits"});

2. db.myCollection.find ({ $where:
   function() { return obj.credits -
   obj.debits< 0; }});
```

If a dynamic threshold value is accepted as user input, the second statement can be rewritten as follows:

```
3. db.myCollection.find ({$where:
   function() { return obj.credits -
   obj.debits< $userInput; }});
```

This may expose a vulnerability where an attacker could overwrite the $userInput variable with arbitrary code, such as

```
$user Input = "0;var date=new
Date(); do{curDate = new
Date();}while(curDate-date< 0;var
date=new Date(); do{curDate = new
Date();}while (curDate-date<10000)"
```

If sanitization check fails to screen the $user Input value, then upon concatenation the third statement becomes the following.

```
4. db.myCollection.find( { $where:
   function() {return obj.credits -
   obj.debits < 0; var date=new Date();
   do{curDate = new
   Date();}while(curDate
   date<10000)";}}};
```

The new query could trigger a Denial of Service (DoS) attack and cause the MongoDB instance to execute at 100% CPU usage for 10 seconds.

5. Security Issue in MongoDB

5.1. Encryption

One of the most serious problems of MongoDB is that data files are never encrypted by default. If encryption is required for the data file, the application layer needs to encrypt the data before sending it to the database server [16].

5.2. Data in Motion (Client-Node Communications)

By default MongoDB does not support SSL client-node communication, which leads to security breach in the Network. To use SSL, it is required recompile whole MongoDB with the "-sl" option or deploy MongoDB enterprise version. Additional steps to generate keys are needed for configuring client/server for SSL communication.

5.3. Authentication

Authentication is disabled by default. Basic MongoDB does provide support for authentication on a per-database level. Users exist in the context of a single logical database. MongoDB Enterprise features an additional Kerberos service for authentication. 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 [8].

5.4. Authorization

Authorization is disabled by default in MongoDB. It provides authorization on a per-database level by using a role-based approach. Available roles are limited to the following: read, readWrite, readAnyDatabase, readWriteAnyDatabase, userAdmin, clusterAdmin, userAdminAnyDatabase, dbAdmin, and dbaAdminAnyDatabase.

5.5. Audit

MongoDB is behind in implementing the desired security logging and monitoring. Most monitoring and reporting tools currently distributed with MongoDB are related to database performance for showing the running state of a MongoDB instance. There is an HTTP Console for each MongoDB instance to show information about the system and connecting clients. However, if security is not enabled for the MongoDB instance, no authorization is needed to access this interface, resulting in a potential vulnerability of audit data leak.

6. Security Issue in Cassandra

6.1. Encryption

The Enterprise edition of Cassandra has come up with the latest feature for Transparent Data Encryption (TDE). This is meant to protect the data being transferred from memory to written disks. Certain extent of this feature is used to protect the sensitive data. However, since the data encryption is stored locally, a secured file system must be used to turn on the TDE. In addition, the commit log of Cassandra, a place where the file is modified to, is not encrypted at all.

6.2. Data in Motion (Client-Node Communications)

Another security issue in the Cassandra is that, by default, the client-node communication is not encrypted. Transmission of the data SSL can be turned on by editing the settings under client_encryption_options in the cassandra.yaml file after generating valid server certificates.

6.3. Data in Motion (Inter Node Communications)

In Cassandra by default the inter-node communication is not encrypted either. If needed, available SSL encryption options include: "all" (allinter-node), "dc" (between datacenters), and "rack" (between racks). Inter-node SSL communication can be configured by editing the corresponding settings under server_encryption_options in the cassandra.yaml file. By default, SSL option is disabled and using default configurations by organizations might lead to data breaches while transmitting information over the network as plain text.

6.4. Authentication

By default the authenticator setting of basic Cassandra is AllowAllAuthenticator, which means there is no authentication. The other available option is Password Authenticator, in which user names and passwords (hashed but unsalted) are stored in the system_auth.credentials table. Enterprise Cassandra can further provide Kerberos authentication, which requires setting up separate Kerberos servers and installing Kerberos client software on all joining Cassandra hosts.

6.5. Authorization

The default choice of Cassandra is AllowAllAuthorizer, which essentially provides no authorization and allows any action by any user. If Cassandra Authorizer is selected, then privileged administrators can grant any of the privileges (ALTER, AUTHORIZE, CREATE, DROP, MODIFY, SELECT) on any resources (ALL KEYSPACES, KEYSPACE, TABLE) to a selected user, by executing CQL.

6.6. Audit

Auditing is available in Enterprise Cassandra's log4j-based integration, and a per-node basis. To get the maximum audit information, turning on auditing on every node is recommended. Filters are available for logging, using a combination of the following categories: ADMIN, ALL, AUTH, DML, DDL, DCL and QUERY.

7. Security Issues in MySQL

7.1. Encryption

MySQL does not support database encryption. However, the encryption can be done at applications level and has provided many inbuilt methods. For making it more secure, MySQL is deployed with strict firewall rules to discard the suspicious packet. It supports SSL and it is not activated by default because of performance issues [9].

7.2. Authentication

MySQL provides several authentication methods such as Old Password Authentication and Secure Password Authentication. The type of method it uses depends upon the CLIENT_SECURE_CONNECTION flag [9]. In Secure Password Authentication, MYSQL uses SHA1 hash function for storing passwords of users. It is more secure when compared to MongoDB and Cassandra.

7.3. Node Communication

With the advancement of cluster computing MySQL was updated in its later releases to provide shared data storage environment for clusters. The data can be transferred between nodes using SSL encryption standards. To be more secure, MYSQL should be deployed with intrusion detection system and some Microsoft security recommendations.

7.4. Auditing

MySQL audit provides an easy to use, policy-based auditing solution that helps organizations implement stronger security controls and satisfy regulatory compliance. Each and every action taken is logged. This includes login and logoff attempts, attempts to access a database or a table, changes to database schema and much more. Some features for auditing include the following.

• Dynamically enable/disable audit stream
• Implement policies that log all or selected login or query activities
• Automatically rotate audit log files based on size
• Integrate XML-based audit log stream with MySQL, Oracle and other third party solutions

8. Attack Examples

Security issues in MongoDB and Cassandra may leads to some types of attacks by exploiting the vulnerabilities. Some of the attacks are commonly noted in MongoDB and Cassandra.

8.1. Attacks on MongoDB

The following attacks were reported in most organizations using in MongoDB.

1) 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. We have provided an

example of this attack in Section 3. With normal SQL injection, a similar vulnerability would allow an attacker to execute arbitrary SQL commands. Since JavaScript is a fully featured language, it allows an attacker to manipulate data and run arbitrary code.

2) DoS Attack: DoS attacks are possible in MongoDB. By default MongoDB does not require authentication and authorization is not configured to work with this attack. An attacker can use valid user credentials and he/she does not have to be an administrator to carry out the attack.

3) XSS Attacks [19]: MongoDB allows developers to write JavaScript for the client and database layer. So inserting arbitrary JavaScript code may bypass security authentication or steal sensitive information.

8.2. Attacks on Cassandra

Following are the attacks possible in Cassandra.

1) CQL Injection: CQL injection attacks are possible with implementation generating dynamic Cassandra queries. This may lead to unauthorized information leakage or modification. Prepared Statements can be used to prevent CQL injection attacks. Alternative approach includes filtering inputs before being processed by an application.

2) DoS: No password is set by default in Cassandra. So attacker may send multiple requests to the system. Then system may crash at a particular point of time through Denial of Service (DoS) attacks. Configuring Authentication mechanism may void such attacks.

3) XSS: Default configuration binds an unauthenticated JMX/RMI interface to all network interfaces, which allows remote attackers to execute arbitrary Java code via an RMI request [10]. This is analogical to well Cross Site Scripting (XSS) attack.

8.3. Attacks on MySQL

Attacks on MYSQL are well known. One of the most important attacks in MYSQL is SQL injection attack. It can be avoided by using prepared statements [15] and some query validations techniques.

9. Conclusion

In the context of MOOC web applications, databases play important role in storing sensitive information. This paper provides a summary of an extensive overview of various vulnerabilities in two NoSQL databases (MongoDB and Cassandra) and SQL (MySQL). Each Database has its limitations and advantages. For example Cassandra supports best Auditing and MongoDB supports Authorization better than Cassandra. Selecting specific databases for MOOC-based web applications would depend on business requirements. The study finds that even a good database is not secure if not configured correctly. MySQL is vulnerable to attacks like SQL injections. Some standards and techniques are proposed to avoid attacks. For NoSQL Databases, there is a need for standards and encryptions to provide secure data storage.

10. References

[1] P. Zikopoulos, C. Eaton, D.deRoos, T. Deutsch, and G. Lapis, (2012). Understanding big data: Analytics for enterprise class hadoop and streaming data. New York, NY: McGraw-Hill.

[2] E. Brewer, (2000, Jun.) Towards robust distributed systems. [Online].Available at http://www.cs.berkeley.edu/ brewer/cs262b-2004/PODCkeynote.pdf (Access Date: March 4, 2017)

[3] S. Gilbert and N. Lynch, “Brewer’s conjecture and the feasibility of consistent, available, partition-tolerant web services,” SIGACT News, vol. 33, pp. 51–59, June 2002.

[4] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber, “Bigtable: A distributed storage system for structured data,” ACM Transactions on Computer Systems (TOCS), Vol. 26, Issue 4, pp. 1–26, June 2008.

[5] E. Brewer, "Pushing the cap: Strategies for consistency and availability,”Computer, Vol.45, Issue 2, 2012, pp. 23-29.

[6] Lior Okman, Nurit Gal-Oz, Yaron Gonen, Ehud Gudes, Jenny Abramov, “Security Issues in NoSQL Databases,” Proc. of 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, pp. 541-547.

[7] L.Yishan and S. Manoharan, “A performance comparison of SQL and NoSQL databases," Proc. of IEEE Communications, Computers and Signal Processing (PACRIM), 2013, pp. 15-19.

[8] Noiumkar Preecha, and Tawatchai Chomsiri, “A Comparison the Level of Security on Top 5 Open Source NoSQL Databases,”Proc. of 9th Inter. Conference on Information Technology and Applications (ICITA), Sydney, Australia.

[9] I. Zoratti, “MYSQL security best practices,” The Institution of Engineering and Technology Conference on Crime and Security, IET, 2006.

[10] L. Okman, N. Gal-Oz, Y. Gonen, E. Gudes, and J. Abramov, “Security issues in nosql databases,” Proc. of10th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), 2011, pp. 541-547.

[11]Coursera. [Online]. Accessed from https://www.coursera.org (Access Date: March 4, 2017)

[12] EdX. [Online]. Accessed from https://www.edx.org (Access Date: March 4, 2017)

[13] Udacity. [Online]. Accessed from https://www.udacity.com

[14] J. Daries, J. Reich, J. Waldo, E. Young, J. Whittinghill, D. Seaton, A. Ho, and I. Chuang, "Privacy, Anonymity, and Big Data in the Social Sciences," ACM Queue, Vol. 12, Issue 7, pp. 1-12.

[15] PHP Prepared Statement and Stored Procedure. [Online]. Accessed from http://php.net/manual/en/pdo.prepared-statements.php (Access Date: March 4, 2017)

[16] MongoDB Security Concept. [Online]. Accessed from http://docs.mongodb.org/master/core/security/

[17] L. Okman, N. Gal-Oz, Y. Gonen, E. Gudes, and J. Abramov, "Security Issues in NoSQL Databases," Proc. of 10th IEEE Inter. Conf. on Trust, Security and Privacy in Computing and Communications, Nov 2011, changsa, China, pp. 541-547.

[18] David Kirkpatrick, Mongodb - Security Weaknesses in a typical NoSQL database, 2013, Accessed from https://www.trustwave.com/Resources/SpiderLabs-Blog/Mongodb---Security-Weaknesses-in-a-typical-NoSQL-database/ (Access Date: March 4, 2017)

[19] OWASP, Cross-site Scripting (XSS). [Online]. Available https://www.owasp.org/index.php/Cross-site_Scripting_%28XSS%29, (Access Date: March 4, 2017)

[20] David Lebron and Hossain Shahriar, "Comparing MOOC-Based Platforms: Reflection on Pedagogical Support, Framework and Learning Analytics," Proc. of 2015 International Conference on Collaboration Technologies and Systems, Atlanta, USA, June 2015, IEEE Press, pp. 167-174.

[21] David Lebron, Hossain Shahriar, Rubana Lupu, "Security concerns and mitigation approaches for MOOC-based applications," Proc. of 10th Int. Conf. for Internet Tech. and Secured Transactions, London, UK, Dec 2015, pp. 145-150.

[22] Cassandra Database. [Online] Available at http://cassandra.apache.org. (Access Date: March 4, 2017)