A COMPARATIVE STUDY OF NOSQL SYSTEM VULNERABILITIES WITH BIG DATA

Kiran Fahd[1], Sitalakshmi Venkatraman[2] and Fahd Khan Hammeed[3]

[1,2]Department of Information Technology, Melbourne Polytechnic, VIC, Australia
[3]Suncorp Group, Melbourne, VIC, Australia

ABSTRACT

With the emerging third wave in the development of the Internet, the past year has witnessed huge data exposure resulting in cyber-attacks that have increased four times as that of the previous year's record. In this digital era, businesses are making use of NoSQL technologies for managing such Big Data. However, the NoSQL database systems come with inherent security issues, which pose a major challenge to many organisations worldwide. There is a paucity of research studies for exposing the security threats and vulnerabilities of NoSQL technologies comprehensively. This paper presents an in-depth study of NoSQL security issues by performing a detailed comparative study of the security vulnerabilities identified in NoSQL database systems. A set of key security features offered by the four commonly used NoSQL database systems, namely Redis, Cassandra, MongoDB and Neo4j are analysed with an aim to identify their strengths and weaknesses. The vulnerabilities associated with built-in security, encryption, authentication/authorization and auditing that impact Big Data management are compared among these popular NoSQL database systems and the risk levels are identified. In addition, illustrations of possible injection attacks experimented with these NoSQL systems are provided. Finally, a high-level framework is proposed for NoSQL databases with considerations for security measures in Big Data deployments. The discussion forms a significant technical contribution for learners, application developers and Big Data deployers paving way for a better awareness and management of the NoSQL systems in an organization.

KEYWORDS

NoSQL; Big Data; security; vulnerabilities; non-relational database; injection attacks

1. INTRODUCTION

Big Data is being used beyond its basic concept of high-volume, high-velocity data. Big Data analytics are being employed by managers to derive deep insights into their businesses [1][2]. Such unearthed knowledge can be used to improve decision-making and business performance. At present, all types of organsiations, small, medium and large, are collecting huge data with the main focus of using Big Data analytics to improve and develop their internet marketing operations for predicting and meeting future consumer demands [3][4]. Organisations have started to realise that the use of Big Data is also important in other business functions resulting in cost reduction and improvements in products and services. However, it is also a huge challenge to manage Big Data as traditional and new risks suggest seven key steps for its success: governance, management, architecture, usage, quality, security and privacy [5][6].

In today's context of Big Data, developments in non-Relational databases (aka NoSQL databases) have achieved the right infrastructure that can be very much well-adapted to support the heavy demands of Big Data [3][4][7]. NoSQL databases are simply schema-less databases capable of handling large amounts of structured and unstructured data like documents, e-mail and multimedia efficiently with flexible data models [8]. NoSQL databases provide high availability,

performance and scalability but in turn trade-off with consistency and security [9]. In the beginning of the twenty-first century when traditional relational database management system (RDBMS) was highly popular, non-relational database started to fill the gaps of RDBMS by providing increase in storage capabilities and performance using horizontal scaling [10]. However, with the application of new methods, unfamiliar security issues also arise. In the world of Big Data, any information leak may mean an incredibly high amount of data has been exposed with a high possibility that it contains sensitive information [11] [12]. Hence, privacy protection is a major issue with NoSQL databases in Big Data environments.

The distributed and cloud computing method adopted by NoSQL databases opens up extra chances of data breaches [6][13]. Organizations have been highly concern about the security breaches and attacks, especially as the frequency and level of severity of security breaches has been doubled over the year. Australia experienced a significant increase of 25.8% in cybercrime costs between 2016 and 2017 as compared to an average increase of 20.4% worldwide [14]. As part of information security strategy, securing the data, which is an extremely valuable asset for any organization, has become a topmost priority. A non-compliance of security and privacy protection of sensitive data adequately can cause significant damage. A database should include the basic security features: Authentication, Access control, System Privilege and Authorization, Auditing, Data Encryption and Network Security [15]. This brings into question the risk level of such security features provided by non-relational databases. There is little discussion on the vulnerabilities and security risks associated with NoSQL technologies. Even though vulnerabilities in NoSQL Database systems is a well-known problem, there is a paucity of research in identifying the vulnerabilities of different NoSQL technologies and their security features comprehensively. This paper takes a modest step to address this research problem by performing an in-depth study of NoSQL security issues and in addition, proposes a high-level framework to help users to deal with NoSQL vulnerabilities.

The rest of the paper is organised as follows. In section 2, we provide the literature background of Big Data and NoSQL databases that lay the platform for the security gaps. Section 2 presents the in-depth study performed on four well-known NoSQL databases (Redis, Cassandra, MongoDB and Neo4j), comparing the main security features and their inherent limitations. In section 4, we illustrate the security weakness in non-relational databases using NoSQL injection attacks. Section 5 describes our proposed high-level framework for NoSQL database security for Big Data application. Finally, section 6 provides conclusions and future research directions.

2. BIG DATA AND NOSQL DATABASES

In Big Data applications, where real-time management of increasing data volumes and diversity is essential, NoSQL databases are considered rather than relational databases to cope with the situation. Though NoSQL databases can address the real-time processing issues of relational databases, with Big Data applications, there are complex security issues, and these require deep understanding in order to address them effectively.

NoSQL databases can be broken down into four major categories as shown in Figure 1: i) Key-Value stores such as Redis, Riak and DynamoDB; ii) Document stores such as CouchDB and MongoDB; iii) Column-oriented stores such as Cassandra and Hbase; and iv) Graph stores such as Neo4j and GraphDB [3][16]. However, some non-relational databases have overlapping characteristics that make them difficult to classify under one of these four categories. Some non-relational databases are a hybrid between two or more non-relational database types. For example, Cassandra could be a Column store or could be a Key-value store as the model is composed of rows containing columns. A key, which is a row id, is required to access the smallest data unit in a column [17]. ArangoDB is a multi-model non-relational database that

satisfies multiple characteristics of the three categories, namely key-value, document, and graph stores [18].

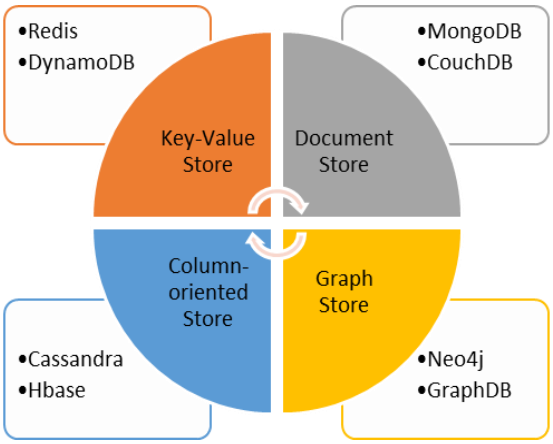

Figure 1. Categories of non-relational databases

Non-relational databases, in general, have not yet achieved full maturity in the security arena [10]. With an exception of Graph databases, all non-relational databases follow the Basically Available, Soft state, Eventually consistency (BASE) design philosophy [19]. BASE satisfies the focus of non-relational databases towards the need to access data quickly and easily, but it adversely impacts the security [8]. The design of non-relational database significantly overlooked new security challenges arising from the complexities of the five V's (Velocity, Volume, Value, Variety, and Veracity) of Big Data. Figure 2 shows the mapping of these Vs to the data acquisition, data storage and data analytics within the four main phases of the Big Data life cycle. The impact on data security during every phase of Big Data life cycle, namely collect, store, analyse and govern is of increasing concern to businesses. In reality, hundreds of computing nodes need to perform parallel database operations in public cloud environments that houses Big Data [20] [21][22].

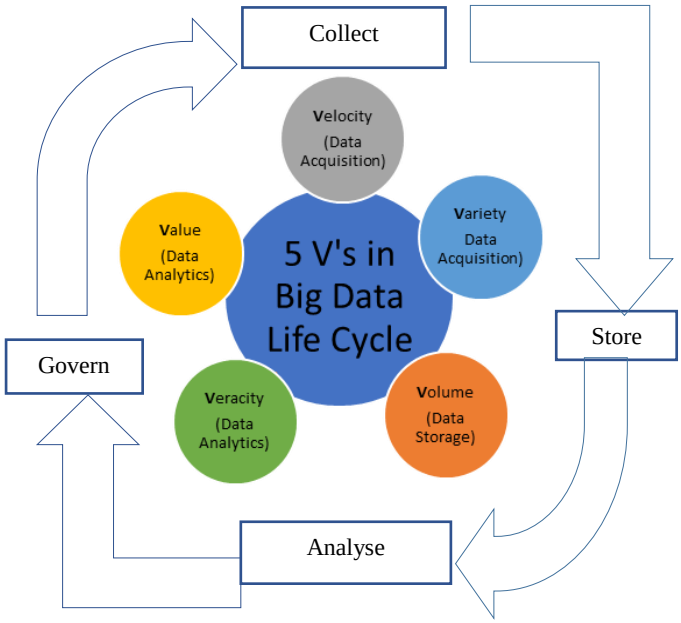

Figure 2. Five V's in Big Data Life Cycle

It is very important to identify the factors necessary for enforcing security in databases and to study the applicability of these factors to every non-relational database equally. The next section discusses the security features and challenges associated with the non-relational databases in view of the basic security features that have an impact on Big Data management.

3. COMPARISON OF NOSQL DATABASE SECURITY FEATURES

We conducted an in-depth comparative study of the key security features by reviewing the existing literature and the vendor documentations on security of non-relational databases such as Redis, Cassandra, MongoDB and Neo4j (one from each of the four major non-relational database categories).The comparisons were made based on key factors covering security features of non-relational databases, namely built-in protection or the default security configuration, encryption, authentication, authorization and auditing for an effective Big Data deployment. Since Big Data applications are prone to injection attacks, we compare NoSQL databases with respect to their injection vulnerabilities. We adopt the assessment criterion comprising various security features for the analysis of sharded NoSQL databases that was proposed in literature [23]. With such a security assessment criterion given in Appendix A, we identify the levels of security metrics (Low, Medium, High) by comparing the security features supported by each non-relational database. While a security level of 'High' refers to a complete set of security features catering to total security of data; 'Medium' refers to restricted security features, or with some features missing; and 'Low' refers to no security or very basic security features. Firstly, we compare the built-in protection supported by each of the four NoSQL databases, namely Redis, Cassandra, MongoDB and Neo4j. Then we compare their levels of basic security features such as encryption, authentication, authorization and auditing.

3.1. Security Feature 1: Built-in Protection

Built-in protection is the integral part of NoSQL database that provides the default security configuration features of the system. We observe that misconfigurations of the operating system (OS) layer, the database layer or the application layer can cause a security threat. Non-relational databases have none or a very weak built-in security layer. Therefore, it depends on external mechanisms like database administration, external security software or application security features to add extensive security features to the non-relational database [13][21]. These databases were not designed with security mechanism as a priority, and therefore, some of the NoSQL database vendors clearly state, in their installation instructions that these databases should be accessed by reliable connections inside trusted environments. Thus, it is not safe for non-relational databases to be directly exposed in an open environment, for e.g. Internet, Web, or the cloud where untrusted clients could directly access NoSQL databases.

The weak security mechanisms of NoSQL databases may cause insider attacks that can remain unnoticed because of inadequate auditing and poor log analysis methods. To overcome these issues on a permanent basis, vendors of non-relational databases have developed a bottom-up security approach since a security breach primarily occurs due to misconfigurations at the OS layer, the database layer or the application layer [8][21]. Table 1 summarizes the security levels of the built-in security configuration features of different non-relational databases for Big Data management.

Table 1. Comparison of NoSQL Security Configuration Features

NoSQL Built-in Security Configuration Features			
Key-Value Store	Redis	Low	Full access to working directory
Column-Oriented Store	Cassandra	Low	No authentication
Document Store	MongoDB	Low-Medium	Basic level auditing
Graph Store	Neo4j	Medium	Supports events logging

Redis Built-in Security Configuration

- Redis is not optimized for maximum security (total protection) due to priority given to maximum performance [24][27].
- It does not support auditing.
- Client full access to Redis working directory
- Attacks have the ability to insert data into Redis that triggers pathological (worst case) algorithm [21][24].

Cassandra Built-in Security Configuration

- Cassandra does not support auditing (inline) and logging in open-source versions
- The *cassandra.yaml* file is the main security configuration file [8][21][17].

MongoDB Built-in Security Configuration

- MongoDB offers basic level auditing of data operations, i.e. writing events to an in-memory buffer [8][21][25].

Neo4j Built-in Security Configuration

- Built-in security is only offered in Neo4j Enterprise Edition
- It supports security events logging e.g. successful or failed user login, in security.log.
- Logging of successful logins is disabled by: *dbms.security.log_successful_authentication=false* query logging in query.log [7][26].

3.2. Security Feature 2: Encryption

Encryption facilitates confidentiality of data and applications in a database including the encryption of data-at-rest and encryption of network data-in-transit to protect Big Data from attackers and unauthorized users [7][8][22]. Encryption is the most efficient way to facilitate data security by converting the data-at-rest and data-in-transit over the network into cipher text. While the relational database encryption is significantly robust and mature, encryption is the weakest feature in NoSQL databases. Thus, in the case of NoSQL databases, anyone having access (authorized or unauthorized) to file system can access the data as summarised in Table 2. Therefore, encryption mechanism at the application and OS layers is required to mitigate this security issue [21][27]. For example, the following Java libraries can be used to encrypt and

decrypt the data before it is persisted to the non-relational database for Java-based Big Data applications or add-ons.

- *import java.security.*;*
- *import java.Crypto.*;*
- *import java.util.prefs.*;*

Table 2. Comparison of NoSQL Encryption Features

NoSQL Encryption Features			
Key-Value Store	Redis	Low	No encryption for data at rest
Column-Oriented Store	Cassandra	Medium	Supported by Enterprise edition
Document Store	MongoDB	Medium	Supports Storage level encryption
Graph Store	Neo4j	Low-Medium	Enterprise edition supports

Redis Encryption

- Redis does not offer data encryption
  - Data-at-rest is stored as plain text
  - Data-in-transit between Redis client and server is not encrypted
- It provides support for SSL proxy (through Spiped) for creating symmetrically encrypted pipes [7][11][24][27].

Cassandra Encryption

- Cassandra does not support data-at-rest encryption in Open-source version; however, it is available in Enterprise Edition.
- Latest version supports SSL encryption between inter-cluster network communication (Node-to-node and client-to-node) [7][8][17][21][23][24][27].
- An example of settings in cassandra.yaml file for node-to-node encryption [17] is given below:

```
internode_encryption: all
keystore: /usr/local/lib/cassandra/conf/server-keystore.jks
keystore_password: myKeyPass
truststore: /usr/local/lib/cassandra/conf/server-truststore.jks
truststore_password: truststorePass
```

MongoDB Encryption

- Data file encryption is not implemented in MongoDB
- It supports SSL-based transport encryption for data-in-transit.
- It supports Storage and Application level encryption and third-party partner plug-ins for data-at-rest encryption [7][8][11][21][22][23][25][27].

- It has introduced at-rest data encryption with a native encryption option of Encrypted Storage Engine for the Wired Tiger storage engine in Enterprise version only [25]. The Encrypted Storage Engine supports many encryption algorithms and further provides safe and secure management of the encryption keys.

Neo4j Encryption

- Neo4j does not support encryption explicitly for data- at-rest.
- It supports the securing of data-in-transit by using TLS/SSL technology which is implemented by Java Cryptography Extension (JCE), a digital certificate and a set of configuration parameters defined in neo4j.conf.
- It provides APIs (OGM) for Java based Application-Level Encryption [7][11][26]. For example,

  class SecureDataConverter implements AttributeConverter {...}

3.3. Security Feature 3: Authentication and Authorization

Authentication is the process of verifying an identity to access the system and authorization is specifying the user's privileges or access to resources [23]. Authentication and authorization are significant issues for the non-relational databases because they do not offer the option to isolate permission to specific Big Data objects [13]. In other words, when users have access to one part of the system, they can access everything without boundaries, making non-relational databases unreliable and vulnerable to malicious attacks.

In order to cut down processing times and to ramp up processor efficiency, the administrator account or authentication is not enabled by default in Big Data environments. In reality, NoSQL databases are not designed with security as a priority. Even installation instructions for few non-relational databases clearly state that non-relational databases should be run on trusted environments only, keeping access to data fast by skipping the authorization and authentication processes. The password storage security provided by the non-relational databases is very weak using authentication techniques such as HTTP Basic, Digest-based authentication and password storage. These weaknesses expose systems to playback attacks, man-in-the-middle attacks or brute force cracking attacks that may result in information leakage. Few non-relational databases provide local-node level authentication. However, these databases do not impose it across all the cluster nodes of Big Data [7][21].

The available authorization solutions in the case of non-relational databases are not powerful and efficient. Different Authorization techniques and solutions are offered by different non-relational databases as summarized in Table 3 [21]. The databases enforce authorization at higher layers (per-database level) instead of enforcing it at lower layers (collection level).

Table 3. Comparison of NoSQL Authentication and Authorisation Features

NoSQL Authentication and Authorization Features			
Key-Value Store	Redis	Low	Only password based
Column- Oriented Store	Cassandra	Low	Role-based internal authentication
Document Store	MongoDB	Medium-High	Client authorization can be enabled
Graph Store	Neo4j	Medium	Allows multiple providers

Redis Authentication / Authorisation

- Redis does not provide authentication by default
- It offers tiny layer of password-based authentication, enabled by AUTH command by system administrators and saved as text in the redis.conf file
- It does not offer authorization [7][11][21][23][24][27].

Cassandra Authentication / Authorisation

- Cassandra stores authentication and authorization information in cassandra.yaml. file.
- No default authentication and authorization. Example parameters from cassandra.yaml file are given below:
  Authorization:AllowAllAuthorizer
  Authenticator:AllowAllAuthenticator
- Weak password-based authentication stored in clear text or as an MD5 Hash, which is not very secure.
- Offers role-based internal authentication by using Cassandra System tables. Example commands for creating and granting roles, and authentication by database using CQL are given below:
  CREATE ROLE admin1 WITH PASSWORD = 'P@$$WORD' AND SUPERUSER =true AND LOGIN = true;
  GRANT testing_admin TO admin1;
  cqlsh -u admin1 -p P@$$WORD1
- An IAuthority interface implements the authorization in Cassandra [7][8][11][17][21][22][23][27].

MongoDB Authentication / Authorisation

- By default, MongoDB supports authentication in standalone or replica-set mode.
- It supports read-only, read-write and admin (DBA and UserAdmin) access.
- It does not support authentication and authorization in sharded cluster.
- It uses SSL with X..509 certificate for intra-cluster authentication.

- Salted Challenge Response Authentication Mechanism (SCRAM) is the default authentication mechanism based on IETF RFC 5802

  mongo --port 27018 -u "AdminUser" -p "passwordf123" --authenticationDatabase "adminDB"

- Enterprise edition supports Kerberos authemtication.
- Authorization is not enabled by default.
- Client authorization can be enabled using --auth or security.authorization setting [7][8][11][21][22][23][25][27].

Neo4j Authentication / Authorisation

- In Neo4j, authentication and authorization is enabled by default. It can be turned off by the setting:

  dbms.security.auth_enabled

- It offers multiple providers to facilitate authentication and authorization.
- In addition to Native, LDAP and custom providers, it supports Kerberos for authentication with single sign-on
- Identity and Access Management (IAM) solution manages access rights and robust permissions [25].

3.4. Security Feature 4: Auditing

Auditing helps in effective Big Data management as it enables audit trail of database activities to track the use of Big Data resources and authority [27]. Auditing is observing to ensure that those without the authorization do not access data and to detect possible password cracking attempts are made before the systems gets compromised. Most of the non-relational databases do not provide auditing features, and this results in a security compromise.

Table 4. Comparison of NoSQL Auditing Features

NoSQL Auditing Features			
Key-Value Store	Redis	Low	Does not support auditing
Column- Oriented Store	Cassandra	Low	No auditing (inline)
Document Store	MongoDB	Low- Medium	Basic level auditing
Graph Store	Neo4j	Medium	Supports security events logging

Redis Auditing

- Redis does not offer any mechanism for auditing [7][11][21][23][24][27].

Cassandra Auditing

- Cassandra does not support any auditing [7][8][11][17][21][22][27].

MongoDB Auditing

- MongoDB offers auditing features in Enterprise version to record audit events, while only basic level auditing of data operations in Open source version.
- By default, the auditing system logs only the authorization failures. To enable the system to log authorization successes, the audit Authorization Success parameter is used [7][8][11][21][22][23][25][27].

Neo4j Auditing

- Neo4j offers limited auditing facilities in Open source and it offers logging facilities in Enterprise edition as given below:
  - Queries executed in the database (*/logs/query.log*) is enabled or disabled by dbms.logs.query.enabled parameter.
  - Security events in the database (*/logs/security.log*) are: login attempts, password change, provision or deprovision of users and roles, enabled and disabled user and unsuccessful attempts by non-admin users to view users and roles
  - debug.log keeps a record of error messages and information useful for debugging issues [25].

- An example of a log file is shown below:

  2018-11-17 14:55:10.766+0000 INFO [Synclogic @ 2017-10-27 ...] [neo4j1]: logged in
  2018-11-17 14:57:43.463+0000 ERROR [AsyncLog @ 2017-10-27 ...] [neo4j2]: failed to log in: invalid principal or credentials
  2018-11-17 14:58:38.556+0000 INFO [AsyncLog @ 2018-10-27 ...] [neo4j1]: created user `neo4j2`
  2018-11-17 14:58:31.553+0000 ERROR [AsyncLog @ 2018-10-27 ...] [neo4j1]: tried to create user `neo4j2`: The specified user ...

4. NoSQL Injection Attacks

NoSQL databases do not support SQL language, and therefore SQL injection attacks are automatically not applicable in Big Data. However, NoSQL databases are very flexible and provide different interfaces and adopt different non-standard query statements that do not incorporate certain security aspects. As a result, NoSQL databases are also vulnerable to injection attacks.

NoSQL injection attacks introduce malevolent code into the query statements embedded at the application level, which is then passed to the database. As a result, the attacker is able to invoke unintended database operations such as CRUD (Create, Read, Update and Delete). The objective of NoSQL injection is to attain unintended behaviour of the query statement by altering query arguments. The attack tries to insert code in the statements and may be successful if user inputs are not validated as shown in Figure 3 [27].

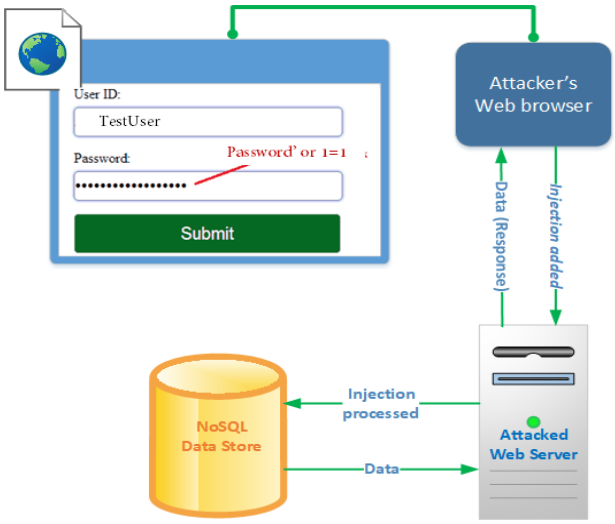

Figure 3. NoSQL Injection Attack

Many non-relational databases like MongoDB and Redis claim that there is no concept of string escaping or building queries with string, therefore, injection is impossible. It is true that the query structure (JSON or Cypher Query Language-CQL) of such NoSQL databases make injection attack difficult, though not impossible as claimed earlier [28][29]. Table 5 shows a comparison of injection attack vulnerabilities with NoSQL databases.

Table 5. NoSQL Injection Attack Vulnerabilities

NoSQL Injection Attack Risks			
Key-Value Store	Redis	Possible with string concatenation	Attack chance with multiple js execution
Column-Oriented Store	Cassandra	Possible with SQLi	CQL parsed like SQL
Document Store	MongoDB	Possible with Java script / string concatenation	JavaScript query arguments/ JSON documents
Graph Store	Neo4j	Possible with string concatenation	CQL using string concatenation

Redis Injection Attacks

- Redis injection attacks are not possible with normal client libraries.
- Redis on Node.js allows multiple different ways to execute the statement creating a chance for injection attack with certain preconditions.
- An example scenario of an injection attack:
  *redis.set(**key**, "default");*

Normal request: sets the JSON body to {key: "level"} and results in *redis.set("level","default");*

Injection request: Instead of sending in string, an array of string will be injected which sets the JSON body to {key : ["level", "high"]} and results in *redis.set(["level","high"],"default")*;

The above commands will be interpreted as an arguments array and will save level=high as a key-value pair and the actual value "default" will be ignored and injection will be successful.

Cassandra Injection Attacks

- Cassandra offers CQL (Cassandra Query Language), SQL-like query language, compatible with the JDBC API. For example: *SELECT deptid* FROM emp *WHERE empid = 104*;
- As CQL is a parsed language like SQL, it is exposed to injection attacks [7][8][21][22].

MongoDB Injection Attacks

- In MongoDB, queries and data are represented in JSON format [30].
- JavaScript, which is an interpreted scripting language, poses potential risk for injection attack [7][8][11][21][22][23][25][27].
- An example of different statements returning the same result from MongoDB [25]:
  - *db.Testdata.find(*
    { age : { $gt: 18 } }
    );
  - *db.Testdata.find(*
    { $where: function() {
    return this.age > 18;}
    }
    );
  - *db.Testdata.find(*
    { $where: "this.age > 18" }
    );
  - *db.Testdata.find(*
    "this.age > 18"
    );

  In the previous two statements, the value in the ‘where’ clause is passed as a string and a concatenated value can be directly passed from the user.
- Another example of a login authentication which takes user input as an argument in JavaScript query is given below. Since userid and passcode are not validated, there is risk of injection attack by injecting JSON document or URL. An attacker can bypass the query by using the following JSON:

  db.user.find({userid: userid, passcode: passcode});

  {
  "userid": {"$ne": ""},
  "passcode": {"$ne": ""}
  }

  Or

  https://mongodbexample.org/login?userid[%24ne]=&passcode[%24ne]=

JSON, $ne means not equal and "$ne": null means userid is not equal to an empty string. The above MongoDB statement with injection of this JSON will become:

db.user.find({ userid: {"&ne": ""}, passcode: {"&ne": ""} });

This will return the user details from the user collection (possibly of an administrator) without comparing its userid and passcode as none of the data in userid or passcode will be null.

Neo4j Injection Attacks

- Neo4j uses Cypher (CQL), which is a declarative graph query language [26].
- Cypher is vulnerable to injection attack by using string concatenation. An example scenario is given below:

START n=node(...)
WHERE n.name = '" + userinput_value +
"'RETURN n

Injection attack can be initiated by changing userinput_value, for example, by giving the userinput value as follows:

"*OR 1=1*
*WITH **
START x=node(*) DELETE x "

The query will delete all nodes in the database, which has a drastic impact.

Overall, since PHP and/or JavaScript engines are employed with sophisticated functions to filter data in NoSQL databases, injection attacks affect the security of the database [31]. In general, an invader familiar with the syntax and data model could perform NoSQL injection attacks with an attack string to be parsed, evaluated, or concatenated into a NoSQL API call. Hence, NoSQL databases possess in distinguishable loop holes and require an appropriate framework that can provides security measures to be adopted while deploying non-relational databases. The aim of the next section is to propose such a framework.

5. PROPOSED FRAMEWORK FOR NOSQL DATABASE SECURITY

Originally motivated by Web 2.0 applications, NoSQL databases that were originally designed to scale simple OLTP-style application loads over many servers, [32] have evolved to become part and parcel of Big Data applications. The use of NoSQL technology was mainly suggested for two reasons: i) to enhance programmer productivity based on application's needs and ii) to improve data access performance in order to handle large data volumes, reduce latency, and increase throughput [33]. Even though it was recommended that most nonstrategic applications, should still use relational databases until the NoSQL ecosystem becomes more mature, the Big Data evolution have driven NoSQL databases to be deployed by all organizations. However, the non-awareness of the above-mentioned security vulnerabilities is having major impact on organisations. Data breaches could occur if NoSQL security vulnerabilities are exploited, and it could result in serious implications. Due to the inherent weak security mechanisms in NoSQL databases, several denial of service (DoS) and injection attacks have been targeted on Big Data systems. As mentioned in previous section, attacks can manipulate a data access language to execute suspicious query statements or make the database server unavailable due to the high volume of illegitimate requests. These types of attacks have been recently reported for MongoDB and Cassandra within Big Data applications [34]. Hence, we propose a framework for considering NoSQL security in Big Data applications.

Without a customized security configuration, a well-designed and best performing database does not offer appropriate data protection. Most of the non-relational databases lack security framework and protocols such as authentication, authorization, and accounting (AAA). Since data is being processed and transmitted in a distributed manner, there are inherent security issues due to insufficient data encryption required for security during the basic information states, such as processing, transmission and storage of Big Data.

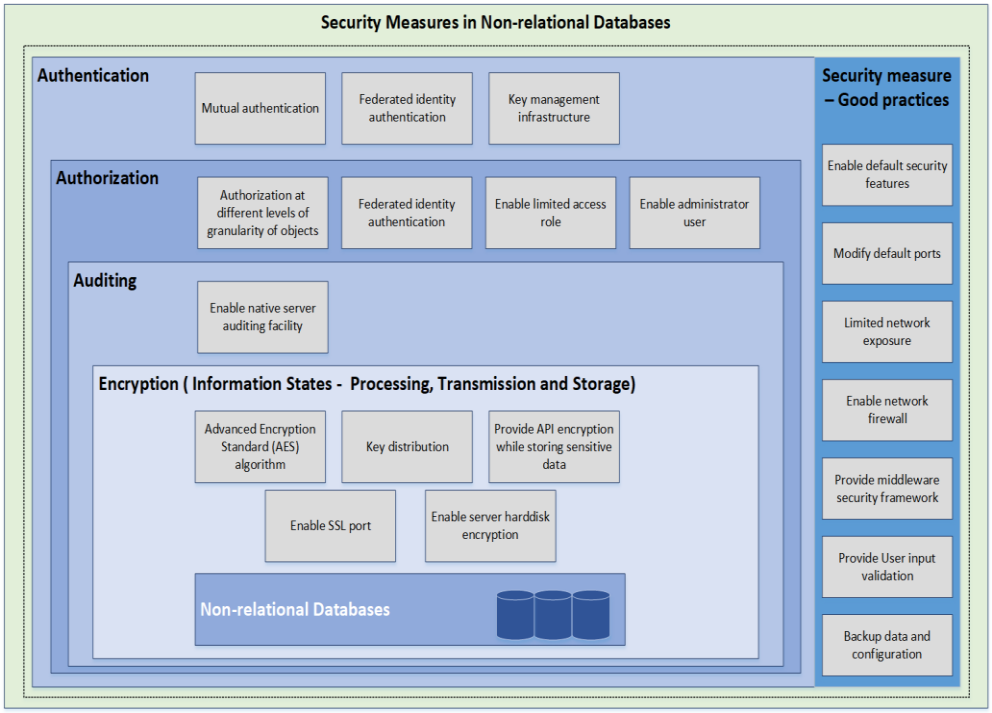

Figure 4. High-level security framework for NoSQL/non-relational databases

We have identified the NoSQL databases are considered to be inherently insecure, and there is a need for better security standards and encryption protocols to provide secure data storage. Previous research studies have identified certain security measures that could help to mitigate the security attacks and lessen the security concerns in Big Data as mentioned earlier [35]. Our in-depth analysis of the currently available NoSQL databases indicates that they offer either none or some security controls depending upon the vendor and license. We provide a high-level framework for security measures that would serve as a guidance for securing NoSQL based Big Data deployments. Figure 4 shows a schematic overview of the various components of our proposed framework. For Big Data applications, non-relational databases are quite suitable to be built for cloud that supports highly scalable structure and architecture. Therefore, the proposed security framework is influenced from the security of cloud environment. The goal is to provide the set of security measures to be considered as good practices in non-relational databases for incorporating the necessary encryption and the AAA protection schemes during every information state such as processing, transmission and storage. For instance, the use of key distribution encryption mitigates the possibility of data alteration. Further, the use of key distribution with federation identity authorization mitigates attacks such as DoS, privilege abuse and injection [35]. By encrypting sensitive data before transit or with encrypted SSL connection, secure transmission or processing of data (data in transit) can be achieved. Similarly, the data in storage state (data at rest) could be secured with the encryption of original data files or by enabling the encryption of the server hard drive.

In object data stores, the network file share is a special case that enables files to be accessed across the network using standard protocols. Hence, appropriate security and concurrent access control schemes are necessary for every read and write operations requested through the distributed services. For instance, the log file in MongoDB is not encrypted within the storage engine and could contain sensitive data. However, recent MongoDB versions have a *security.redactClientLogData* setting which prevents any potential sensitive data from being saved into the process log file. On the other hand, since this setting affects the log diagnostics, appropriate encryption schemes for the log files are also recommended.

Encryption also comes with a performance trade-off as it results in increased latency in Big Data due to the huge volume of documents involved. Therefore, there is a need to protect sensitive information without degrading the performance and scalability of the NoSQL based Big Data systems. While read operation performance degradation due to encryption could be as low as 5%, write operations could experience a latency of as high as 20%. Hence, new storage engines are devised for achieving the desired performance, security, and scalability by adopting page-level encryption based on the need instead of encrypting the entire database. However, third-party encryption schemes could hamper these settings and hence our framework proposed would assist in setting security standards by different organizations that can be complied while adopting the encryption procedures to secure data both at rest and in motion.

6. CONCLUSIONS AND FUTURE WORK

There is a significant increase in the use of non-relational databases for Big Data management. However, the lack of awareness of the associated security risks are of growing concern as cyber breaches are on the rise. Non-relational databases are relatively young and continuously improving with Big Data requirements. Despite many security improvements done by vendors of NoSQL database platforms, the escalating cyber-attacks call for an in-depth understanding of their vulnerabilities to significantly improve their security mechanisms. There is a paucity of research in discussing the security vulnerabilities of NoSQL databases and the way forward for considering security measures for Big Data applications. This paper took a modest step to fill the gap by performing an in-dept study of the NoSQL database security problem and proposing a framework for undertaking NoSQL security measures. The main contributions of this paper are three-fold. Firstly, this paper discussed the critical security flaws in four different non-relational databases, each database belonging to a different non-relational database category. The security comparison showed that different NoSQL databases offer different levels of security. Secondly, this paper conducted a technical evaluation of the NoSQL databases with respect to their security feature drawbacks. The vulnerabilities of NoSQL databases and possible injection attacks were illustrated with each of the popular NoSQL databases to add value in the context of Big Data applications. Thirdly, the in-depth study resulted in the proposal of a high-level security framework for non-relational databases with the required recommendations for encryption and AAA protection schemes during every information state such as processing, transmission and storage of Big Data life cycle.

In the coming years, since malicious attacks and defenses would evolve, future research is required to conduct studies in the domain of non-relational databases with an objective to design and develop a holistic security framework. The framework will be aimed not only to address the security flaws but more importantly, standardise the security features applicable to each category of NoSQL databases, without losing their scalability and performance characteristics. Our future work would involve hybrid model-based security framework in order to overcome individual weaknesses of NoSQL databases and for arriving at a collective policy in achieving high security levels.

ACKNOWLEDGEMENTS

The authors wish to acknowledge the industry support to verify the NoSQL commands used in the paper that are commonly employed in practice.

Appendix A

TABLE 1. SECURITY ASSESSMENT CRITERIA
(Zahid, Masood and Shibli, 2014)

Authentication and Authorization
High
▪ Sharded database cluster should provide one of the following authentication mechanisms such as
- Two-factor authentication (e.g. passwords with fingerprints, PIN code with mobile number etc.)
- Certificate based authentication using PKI with LDAP
▪ It should also support client side, intra-cluster as well as inter-cluster authentication.
▪ Dynamic access control rules following principal of least privileges, separation of duties with custom defined access control policies
▪ such as UCON (Usage Control Mode) etc.
Medium
▪ Supports only one type of authentication e.g. SSO, OpenID, SAML or some certificate based authentication etc. either within a sharded
▪ cluster or with client.
▪ Support for either RBAC , DAC, MAC, ABAC or FGAC etc.
Low
▪ Only Simple Password based client authentication is supported or no authentication at all.
▪ Few predefined roles such as user and DBA or no other support for access control at all.
Built-in Security
High
▪ This type of database configuration security ensures
- Security of Database Log and configuration files
- Hardened database servers
- Limited number of protocols, services and administrator accounts
- Secured backup and recovery of databases
- Regular Patching and updates etc.
Medium
▪ Only supports a limited number of protocols for communication and strict access control to registries and accounts
Low

▪ Use of default configurations
Encryption
High
▪ Encryption of data-at-transit as well as data-at-rest at application, network and database level. Use of secure protocols like SSL, TLS etc.
and strong encryption with secure key management mechanisms for data security.
Medium
▪ Encryption of data-at-rest or use of transport layer security
Low
▪ No security of data-at-rest or data during transmission
Auditing
High
▪ Transparent auditing of database, application and user profiles using auditing and monitoring tools. Logging of all internal and external activities and events.
Medium
▪ Database level auditing, logging of all changes to user profiles
Low
▪ Database connection level auditing (i-e log-on, log-off etc.) or No auditing mechanisms

REFERENCES

[1] Cao, L. Data Science and Analytics: A New Era, International Journal of Data Science and Analytics, 2016, 1(1), pp. 12.

[2] Mukherjee, S. and Shaw R. Big data concepts, applications, challenges and future scope, International Journal of Advanced Research in Computer and Communication Engineering, 2016, 5(2), pp. 66-74.

[3] Venkatraman S., Fahd S., Kaspi S.,and Venkatraman R. SQL Versus NoSQL Movement with Big Data Analytics, International Journal of Information Technology and Computer Science(IJITCS), 2016, 8(12), 59-66.

[4] Yang, C., Huang, Q., Li, Z., Liu K. and Hu, F. Big Data and cloud computing: innovation opportunities and challenges, International Journal of Digital Earth, 2017, 10(1), 13-53.

[5] Baccam, T. Making Database Security an IT Security Priority, SANS Whitepaper, SANS Institute InfoSec Reading Room, 2009.

[6] Gholami A. and Laure E. Security and privacy of sensitive data in cloud computing: a survey of recent developments, Computer Science & Information Technology (CS & IT)-2016, 2016.

[7] Grolinger K., Higashino WA., Tiwari , A. and Capretz, AM. Data management in cloud environments: NoSQL and NewSQL data stores, Journal of Cloud Computing: Advances, Systems and Applications, 2013, 2:22.

[8] Okman, L., Gal-Oz, N., Gonen, Y., Gudes, E. and Abramov, J., Security Issues in NoSQL, TrustCom IEEE Conference on International Conference on Trust, Security and Privacy in Computing and Communications 2011, 7, pp. 541-547.

[9] Wu L, Yuan L. and You J. Survey of large-scale data management systems for big data applications. Journal of Computer Science and Technology, 2015, 30(1), pp. 163–183.

[10] Mohamed, M, Altrafi, O., and Ismail, M. Relational vs. NoSQL Databases: A Survey, International Journal of Computer and Information Technology, 3(3), 598-601.

[11] Sahafizade, E. and Nematbakhs, M. A Survey on Security Issues in Big Data and NoSQL, ACSIJ Advances in Computer Science: an International Journal, 2015, 4(16), 2322-5157.

[12] James B.E. and Asagba P.O. Hybrid Database System for Big Data Storage and Management, International Journal of Computer Science, Engineering and Applications (IJCSEA), 2017, 7(3/4), pp. 15-27.

[13] Cobb, M. NoSQL security: Do NoSQL database security features stack up to RDBM?, Techtarget, 2018.

[14] Online Trust Alliance, Cyber Incident & Breach Trends Report, Internet Society, 2018.

[15] Malik M. and Patel, T. Database Security - Attacks and Control Methods, International Journal of Information Sciences and Techniques (IJIST), March 2016, 6(1/2), pp. 176-183.

[16] Deepak, GC. A Critical Comparison of NOSQL Databases in the Context of Acid and Base, Culminating Projects in Information Assurance. 2016, Paper 8.

[17] Apache Software Foundation. Apache Software Foundation: The Apache Cassandra Project, Apache.org, 2018.

[18] ArangoDB GmbH. What is a multi-model database and why use it?, ArangoDB White Paper, December 2016.

[19] Chandra D., BASE analysis of NoSQL database, Elsevier Future Generation Computer Systems, 2015, 52(11), pp. 13-21.

[20] Chen, Z. K., Yang, S. Q., Tan, S., Zhao, H., He, L., Zhang, G. and Yang H. Y. The Data Allocation Strategy Based on Load in NoSQL Database, Applied Mechanics and Materials, 2014, 513–517, pp. 1464–1469.

[21] Dadapeer, Indravasan, N. and G. A Survey on Security of NoSQL Databases, International Journal of Innovative Research in Computer and Communication Engineering, 2016, 4(4), 5249-5254.

[22] Kapadia Gayatri S. and Morena Rustom D., Comparative Study of Role Based Access Control in Cloud Databases and NoSQL Databases, International Journal of Advanced Research in Computer Science, May-June 2017, 8 (5), pp. 51-57.

[23] Zahid, A. , Masood, R. & Shibli, A. Security of sharded NoSQL databases: A comparative analysis. Conference Proceedings - 2014 Conference on Information Assurance and Cyber Security (CIACS), 2014, pp. 1-8.

[24] Redis, 2018, https://redis.io/topics/security

[25] MongoDB, MongoDB Security Architecture, MongoDB White Paper, 2018.

[26] Neo4j, Security, Neo4j Operations Manual - Chapter 9, 2018.

[27] Noiumkar P. and Chomsiri T., A Comparison the Level of Security on Top 5 Open Source NoSQL Databases, presented at the 10th International Conference on Information Technology and Applications; Sydney, Australia. 2014.

[28] Ron, A., Shulman-Peleg, A., and Bronshtein, E., No SQL, no Injection?, IEEE S&P workshop on Web 2.0 Security & Privacy, 2015.

[29] Shahriar, H. and Haddad, H.M. Security Vulnerabilities of NoSQL and SQL Databases for MOOC Applications, International Journal of Digital Society (IJDS), 2017, 8(1), pp. 1244-1250.

[30] Chodorow K, Dirolf M. 2010. MongoDB: the definitive guide. Sebastopol: O'Reilly Media.

[31] Ahmad, K.,.Alam M.S and Udzir N.I. Security of NoSQL Databases Against Intruders, Recent Patents on Engineering, 2019, 13, pp. 1-8

[32] Cattell, R. Scalable SQL and NoSQL data stores. ACM Sigmod Record 39(4), 2011, pp. 12-27.

[33] Sadalage, P.J. and Fowler, M. NoSQL distilled: A brief guide to the emerging world of polyglot persistence. Pearson Education, 2013.

[34] Shahria, H, and Hadda, H. Security Vulnerabilities of NoSQL and SQL Databases for MOOC Applications, International Journal of Digital Society (IJDS), Volume 8 (1), 2017.

[35] Ghazi Y, Masood R., Rauf A., Shibli A., and Hassan O. DB-SECaaS: a cloud-based protection system for document-oriented NoSQL databases, EURASIP Journal on Information Security, 2016, 2016: 16.

AUTHORS

Kiran Fahd received the B.Eng in software engineering and the Master degree in Enterprise Planning System. Since then, she has worked in the capacity of Software Engineer and held lecturing positions in Australian University and overseas. She has a number of publications and her research interest are Big data and non-relational databases. Currently, she is a PhD student researching on an innovative creation of a graph-based repository

Dr. Sitalakshmi Venkatraman earned her PhD in Computer Science. She has more than 30 years of work experience both in industry and academics within India, Singapore, New Zealand, and in Australia. She specialises in applying efficient computing models and data mining techniques for various industry problems and recently in the e-business, e-security and e-health domains. She has published nine book chapters and more than 150 research papers in internationally well-known refereed journals and conferences. She is a Senior Member of professional societies and editorial boards of international journals and serves as Program Committee Member of several international conferences.

Fahd Khan has Master degree in Information Technology and is an Enterprise Architect with 20 years of experience leveraging in-depth technical acumen to provide creative business solutions. Experienced in strategic planning, guiding technology architecture and design, and involved in architecture of highly scalable, secure, distributed applications, aligned with company standards, process, methodologies, and best practices. He is interested in research/teaching in information security and high performance computing.