A Comparative Analysis of Security Features and Concerns in NoSQL Databases

Evans Ankomah[1], Charles Roland Haruna[2](✉), Francis Xavier Kofi Akotoye[2], Brighter Agyemang[3], Kwame Opuni-Boachie Obour Agyekum[4], Alexander Asante[5], Lawrence Ephrim[2], and Alexander N. T. Kissiedu[1]

[1] Information and Technology Training and Support Section, University of Cape Coast, Cape Coast, Ghana
{evans.ankomah,alexander.kissiedu}@ucc.edu.gh

[2] Department of Computer Science and Information Technology, University of Cape Coast, Cape Coast, Ghana
{charuna,fakotoye}@ucc.edu.gh

[3] School of Computing and Information Technology, University of Wollongong, Wollongong, Australia
brighter@uow.edu.au

[4] Department of Telecommunication Engineering, Kwame Nkrumah University of Science and Technology, Kumasi, Ghana
kooagyekum@knust.edu.gh

[5] Network and Infrastructure Section, University of Cape Coast, Cape Coast, Ghana
alex.asante@ucc.edu.gh

Abstract. Current developments in cloud computing and also distributed internet applications have produced the demand to store huge quantities of data in dispersed databases that offer high availability and scalability. Over the last few years, increasingly more firms are adopting various sorts of non-relational data sources, frequently described as NoSQL data sources. By definition, NoSQL systems are not relational and also do not provide full SQL capabilities. Moreover, unlike relational databases, it uses consistency and security in exchange for performance and scalability. Security issues end up being more important as a growing number of sensitive data is kept in NoSQL databases.

This article, therefore, examines the security features in four of the most preferred NoSQL databases; Cassandra, MongoDB, Redis and Neo4j, one from each category of databases. Unlike in most survey literature on NoSQL security features, that comparisons made excluded the graph database Neo4j, it is inclusive in this work, which makes this survey unique from others. Also, this will give awareness to developers and database administrators and also help them in choosing the best platform for deploying NoSQL databases.

Keywords: NoSQL · Database security · Comparative analysis

E. Ahene and F. Li (Eds.): FCS 2022, CCIS 1726, pp. 349–364, 2022.
https://doi.org/10.1007/978-981-19-8445-7_22

1 Introduction

A NoSQL database also referred to as “Non-SQL” or “Not Only SQL,” is a database which stores data in various formats other than relational tables. NoSQL databases were introduced in the late 2000s when the cost of storage drastically reduced, and are now one of the buzzwords of modern data storage systems. This is due to the large amount of data that currently exists and the swiftly growing heterogeneous data sources like sensors, GPS as well as several other types of smart gadgets. Web 2.0 companies (such as Amazon Facebook and Google) are the main drivers of NoSQL databases due to their increasing data and infrastructure demands. Web 2.0 has brought about numerous new apps that rely on the storage and also processing of large amounts of data and require high availability and scalability. This poses additional challenges for RDBs [1]. Primarily, the objective of these databases is to distribute large amounts of data across many cloud servers. There is growing interest in efficiently processing this unstructured data, commonly referred to as «big data »and incorporating it into traditional applications. But recently, NoSQL databases have built-in protection mechanisms to prevent security attacks [2–4].

This white paper analyzes the security features and issues of the four most preferred NoSQL databases-one from each of the four major categories [45] of NoSQL databases [44]. Specifically, Cassandra (column-based database), MongoDB (document database), Redis (key-value store), and Neo4j (graph database). It describes the key security features and issues of these four database systems under vulnerability to authentication, authorization, communication encryption, auditing, and DoS/injection attacks. In most survey literature on NoSQL security features, the comparisons made did not include the graph database Neo4j, which makes this survey unique from others.

2 Overview

2.1 Cassandra

Cassandra is an open-source, distributed, and also decentralized storage system (database) that offers an extremely readily available solution without a single point of failure [5]. Cassandra is a column-oriented database that is consistent, fault-tolerant, as well as scalable, and it runs on a network of hundreds of nodes. Its replication model is based on Amazon’s Dynamo [6], and its data design is based on Google’s BigTable “column family” data model [7]. Cassandra is thus a hybrid data management system that combines a column-oriented DBMS (e.g., Bigtable) and a row-oriented store. Cassandra was designed to work with Facebook’s Inbox Search feature [5]. Cassandra combines BigTable’s data structures with Dynamo’s high availability to serve over 100 million users daily. Cassandra combines BigTable’s data structures with Dynamo’s high availability to serve over 100 million users daily. Currently, Twitter, Facebook, Cisco, eBay, Rackspace and Netflix are some of the largest companies using this type of database The following are some of the several outstanding features of Cassandra [5,8–12].

2.2 MongoDB

MongoDB is a document database created by 10gen that was designed for ease of development and scaling [13]. Written in C++, MongoDB is a schema-free, document-oriented database that manages JSON-like document collections. This enables Data to be nested in sophisticated hierarchical structures while remaining queryable and indexable. As a result of this, it allows many applications to model data more naturally. MongoDB makes use of collections and documents instead of rows and tables as in relational database management systems (RDBMS). Some of MongoDB's key features are as follows: Scalability/sharding, MongoDB Query Language, Indexing, Data Replication and Document Oriented nature [14–18].

2.3 Redis

Remote Dictionary Server, widely recognized as Redis, is a fast, open-source, in-memory key-value data store. It is an in-memory NoSQL database that recognizes a variety of data structures, including strings, lists, sets, hashes, as well as sorted sets. Salvatore Sanfilippo wrote Redis in C Language and released it in 2009. Currently, Redis provides sub-millisecond latency, allowing for countless thousands of transactions per second in real-time applications such as gaming, healthcare, advertising and others. Redis, unlike the other key-value stores, provides data structures for handling any form of binary data, including arrays, bytes, numbers, strings, XML documents, images, and so on [19]. Furthermore, Redis provides hashes for storing and querying the database's objects. Listed here are some of the features of Redis database: Scalability and High availability, In-memory performance, Replication and persistence and Rich Data Structures [20–26].

2.4 Neo4j

Neo4j is arguably the most popular open-source Graph Database in the world. Written in Java, Neo4j adheres to a data model known as the native property graph model. The graph contains nodes (entities), which are linked together (by relationships). Data is stored in key-value pairs known as properties by nodes and relationships. Each piece of data is explicitly linked, resulting in unprecedented speed and scale. Neo4j is powered by a native graph database which stores and manages data in a more natural and connected manner, allowing for ultra-fast queries, a deeper context for analysis, and easily modifiable data relationships. Some of it's Key features includes: Reliability and Scalability, Data model, Cypher Query Language and Indexing [27–30].

Summarized in Table 1 are the features of MongoDB, Casandra, Redis and Neo4j; the NoSQL databases presented in this paper.

Table 1. Features of presented NoSQL databases.

Database features	NoSQL databases			
	Casandra	MongoDB	REDIS	Neo4J
Cypher query language				✓
Data model				✓
Document oriented		✓		
Elastic scalability	✓			
Fast writes	✓			
Fault tolerance	✓			
Indexing		✓		✓
In-memory performance			✓	
Peer to peer decentralized architecture	✓			
Query language	✓	✓		
Reliability and scalability				✓
Replication and persistence		✓	✓	
Rich data structures			✓	
Scalability/sharding and high availability		✓	✓	
Tunable consistency:	✓			

3 Security Features in Cassandra, MongoDB, Redis and Neo4j

Security has been a weakness in all NoSQL databases. There is no NoSQL database that provides complete security. As stated previously in the introduction, the primary concern of NoSQL Database designers was not based on security; hence, there are numerous security concerns in their design. This section focuses on some of the security features and issues with Cassandra, MongoDB, Redis, and Neo4j. Specifically, this section will look at how secure these systems are against vulnerabilities to authentication, authorization, communication encryption, auditing and DoS/Injection attack criteria [43] and possibly outline briefly the main issues in each one.

3.1 Cassandra Security Features

1. Authentication:
   Cassandra supports pluggable authentication and it is configured via «the authenticator »settings in «cassandra.yaml ». In Cassandra's default distribution, there are two choices available. This implementation removes the need to authenticate to the database and is thus used to completely disable authentication. The «Password Authenticator »is another option, in which usernames are hashed but unsalted MD5 passwords [31] are saved in the system's «auth.credentials table ». To manage security in enterprise Cassandra,

you can also utilize external, third-party packages such as Kerberos authentication. This will necessitate the installation of separate Kerberos servers as well as Kerberos client software on all joining Cassandra hosts.

2. Authorization:
   Similar to Authentication, Cassandra also supports pluggable authorization, which is customizable via the «authorizer »setting inside «cassandra.yaml ». It also comes with two major options to choose from. The default distribution enabled is the «AllowAllAuthorizer ». This undertakes no checks and thus provides no authorization; hence it gives full permissions to all users irrespective of their roles. The second choice is the «CassandraAuthorizer ». This provides full permissions management capability and saves its data in «Cassandra's system tables ». By selecting this option, privileged administrators gain the ability to enable any of the privileges on any resource to a selected user by running the CQL. The problem with the Cassandra Authorizer approach is its inability to refresh the file on each access, making it impossible to modify the valid permissions without restarting the entire Cassandra process.
3. Communication encryption:
   Encryption in Cassandra is transparent to all end-user activity. You can read, insert, update, etc. data without changing anything on the application side. Cassandra comes with multiple levels of encrypting data such as auxiliary encrypted mode of communication (client node communication) from the "client machine" to the "database cluster". By default, client node communication is unencrypted, but can be enabled after a valid server certificate is generated. The Client-to-server SSL ensures that data in flight is not compromised and that client machines are securely transferred back and forth. Consequently, "node-to-node encryption" can be used to make sure that data is secured as it is transferred between database cluster nodes. This can also be customized by changing the appropriate settings in "server_encryption_options" in the "cassandra.yaml" file. The SSL feature is deactivated by default, because using these default settings by firms may result in data breaches while sending data over the network in plain text. Finally, in DataStax Enterprise, transparent data encryption (TDE) prevents "data at rest" from theft and unlawful use [32]. Because the data encryption is kept locally, the TDE must be enabled when using a secure file system. Similarly, Cassandra's commit log (the location where the file is edited) is also not secured.
4. Auditing:
   Cassandra 4.0 and higher versions comes with audit Logging [33]. This is used to log all incoming CQL command requests, as well as authentication to a Cassandra node. In the cassandra.yaml file, the custom logger can be implemented and injected with the class name as a parameter. An administrator can use data auditing to determine "who looked at what/when" and "who changed what/when". However, executing prepared statements in Cassandra will log the query as provided by the client in the prepare call, along with the execution timestamp and all other attributes.

5. Vulnerability to DoS/Injection Attack:
   Cassandra utilizes a "Thread Per-Client" approach in its network code. With this, an attacker can prevent the Cassandra server from accepting new client connections by causing the Cassandra server to allocate all its resources to fake connection attempts. However, Cassandra offers creating user-defined-functions (UDFs) functionality to perform custom processing of data in the database. But JFrog's Security Research team [34] recently disclosed a remote code execution vulnerability that they said is "easy to exploit and has the potential to wreak havoc on systems." This is possible because even though these new vulnerabilities do not affect Cassandra default installations where UDFs are disabled, many Cassandra configurations enable them, causing the instance to be vulnerable to DoS attack.

3.2 MongoDB Security Features

1. Authentication:
   Enabling authentication is essential for MongoDB security because it is not enabled by default. Since MongoDB does not have a distinct user directory, authentication data is kept as part of MongoDB databases. By default, MongoDB employs the Salted Challenge Response Authentication Mechanism (SCRAM) when authentication is enabled. The IETF RFC 5802 standard provides the foundation of this system. With a customizable iteration count and unique random salts for each user, it allows for bi-directional authentication between client and server. It is compatible with both SHA-1 and SHA-256 hashing. In addition, MongoDB has other authentication options such x.509 certificate authentication, Kerberos authentication, Microsoft Active Directory authentication, and Lightweight Directory Access Protocol (LDAP) authentication [35]). Members of replica sets and sharded clusters can use the x.509 certificate authentication for client authentication as well as internal authentication. However, a secure TLS/SSL connection is required to authenticate x.509 certificates. In this situation, MongoDB's authentication feature needs to be active so that each server may be verified before entering the cluster [36].
2. Authorization:
   Similar to the authentication described, MongoDB authorization is not enabled by default. You can enable authorization by using "-auth" or "security.authorization" setting [3,37]. You can also enable internal authentication for client authorization. Once MongoDB authorization is enabled, it allows to set permissions that are either explicitly assigned to a role, inherited from another role, or both. You can use the default database roles, or specify new roles if they are insufficient for your purposes. MongoDB also utilizes Role-Based Access Control (RBAC) to regulate access to the system. If a user is assigned one or several roles based on which resources and operations you want the user to perform. But aside from the role assignments, users have no access to the system. Also, MongoDB version 3.4 and higher versions support LDAP authorization, which allows the authenticated user to query the LDAP

server to know the LDAP groups it belongs to. MongoDB links the Distinguished Names (DN) of every corresponding group with roles in the admin database. Following this, the user can then be authorized by MongoDB based on the linked roles and privileges.

3. Communication encryption:
   MongoDB encryption provides robust features, some of which are pre-installed on the MongoDB Atlas Data-as-a-Service platform. MongoDB Atlas includes client-to-server TLS encryption as a requirement. MongoDB's "encryption at rest" is an Enterprise functionality that needs Enterprise binaries to provide a layer of security to ensure that written files or storage are only visible after they have been decrypted by an authorized process/application. MongoDB version 4.2 also provides "encryption in use". This allows MongoDB Clients such as drivers and shell to instantly encode and decode fields using secure keys stored in a secure vault.
4. Auditing:
   For mongod and mongos instances, MongoDB Enterprise has an auditing feature. This auditing facility allows administrators and users to track system activity for deployments with multiple users and applications [38]. To enable audit logging in MongoDB, you need to go to the mongod.conf configuration file. The auditing system, when enabled can record the operations of the schema, replica set and sharded cluster, authentication and authorization, and CRUD operations. MongoDB Atlas also provides support for auditing all M10 and larger clusters.
5. Vulnerability to DoS attack:
   MongoDB by default does not enforce authentication as already stated. In many instances, this can allow anyone on the network to access all data within the database. This leaves MongoDB vulnerable to DoS attacks. An attacker does not need to be an administrator to conduct the attack; because they can use any legitimate user credentials.

3.3 Redis Security Features

1. Authentication:
   Even though Redis doesn't attempt to provide access control, it offers a thin layer of optional authentication that may be activated by modifying the redis.conf file. Redis versions before Redis 6 were only able to understand the one-argument version of the command: AUTH. In this configuration, unless the connection is authenticated by AUTH, Redis will reject any command issued by newly connected clients. In Redis 6, it is possible to use the AUTH command in two-arguments form: AUTH. This technique, however, provides backwards compatibility. Additionally, the AUTH command, like all other Redis commands, is delivered in clear text and is not secure against eavesdropping by an intruder with sufficient access to the network.
2. Authorization:
   Redis comes with an Authorization layer when installed. Once the authorization layer is enabled, any query from an unauthenticated client will be

rejected by Redis. A client can authenticate itself by sending the "AUTH" command preceded by the password provided by the system administrator in clear text inside the Redis.conf file. Although a strong password can be generated using the ACL GENPASS command, hackers can take advantage of Redis' great performance to test many passwords simultaneously in a short amount of time. Also, you would have to restart your Redis server after editing the configuration file.

3. Communication encryption:
   Redis does not by default support any form of encryption. Redis does not support SSL-encrypted connections because it's been created for usage only in trusted private networks. Assuming that encryption is desired in the client-server connection, extra tools are necessary. It does not offer data encryption for Data-at-rest (stored as plain text) and Data-in-transit between Redis client and server is not encrypted. Redis, therefore, uses stunnel to encrypt Redis communication. It is an SSL encryption wrapper between a local client and a local or remote server. This stunnel application can tunnel unencrypted communication via an encrypted SSL tunnel to another server [39]. Although SSL encryption is added by stunnel, this does not completely ensure that unencrypted communication will never be recorded. Any attacker will be able to intercept unencrypted local communication as it is being transmitted to Stunnel if they can breach the server or client-server relationship.
4. Auditing:
   Redis has service logs that compile and document operations taken on various Redis entities. The account itself, users, API Keys, subscriptions, databases, accounts, payment methods, and more are examples of these entities. Syslog and local text log files are the two mechanisms that Redis offers for logging. Syslog takes in log messages, directs them to different on-disk log files, and takes care of rotation and deletion of old logs. This method of logging files can present problems because numerous services are writing to numerous log files.
5. Vulnerability to DoS/Injection Attack:
   Redis is an open-source, in-memory database that persists on disk as already indicated earlier on. By default, Redis can be accessed without credentials and can be exploited to corrupt the heap and potentially result in remote code execution. DoS attack is a key threat that Redis does not address. This attack is possibly done by inserting elements into the input set and changing a constant time-taking algorithm to a linear or exponential time-taking method. This will render the system inoperable, resulting in the Distributed denial of service attack.

3.4 Neo4j Security Features

1. Authentication:
   Neo4j make use of user details such as username and password. Passwords are encoded using the SHA-256 format. It has an authentication

module that utilizes the AuthenticationPlugin interface. In Neo4j, authentication is enabled by default but can be turned off by the setting using dbms.security.auth_enabled. It includes a "native auth provider" that keeps the users and their role information in the database. In addition to the Native auth, LDAP auth Provider is also available. Similarly, Neo4j also provides "Single Sign-On" provider and "Custom-built" plugin auth providers for clients with special requirements that are not handled by either native or LDAP. Again, Neo4j supports Kerberos for authentication with single sign-on.

2. Authorization:
   Similar to the authentication, authorization is enabled by default in Neo4j. It comes with the authorization module which utilizes the AuthorizationPlugin interface. Neo4j connects data along with intuitive relations to make identity and access management happen quickly and effectively. Neo4j 3.1 introduced the concept of role-based access control (RBAC). This allows you to possibly create users and grant them specific roles in the database. This was enhanced significantly in Neo4j 4.0 with the inclusion of privileges. However, it is impossible to have different security privileges on different instances of a cluster [40]. As the whole cluster shares the privileges already configured in the database using Cypher administrative commands. This indicates that consumers have the same privileges irrespective of the server they access inside a cluster.
3. Communication encryption:
   Neo4j does not currently deal with encryption for data-at-rest explicitly [41]. However, 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 options provided in neo4j.conf. The SSL framework supports using common SSL/TLS technology to secure the following Neo4j communication channels [42]. Neo4j also provides APIs (OGM) for Java-based Application-Level Encryption [3,37].
4. Auditing:
   Neo4j offers limited auditing facilities in Open source and it offers logging facilities in Enterprise. The systems root directory where the general log files are stored can be configured via "dbms.directories.logs". Queries executed in the database can be enabled or disabled by dbms.logs.query.enabled parameter. Neo4j includes security event logging, which logs all security events. It records login attempts, authorization failures from role-based access control and all administration commands and security procedures that run towards the system database.
5. Vulnerability to DoS/Injection Attack:
   Noe4j prevents cypher injection by sending input as a parameter to the query. In a parameterized query, placeholders can be used for parameters and their values supplied at execution time. This means developers do not have to resort to string building to create a query. Moreover, parameters greatly simplify Cypher's caching of execution plans, resulting in quicker query execution times. Parameters can be used for, (literals and expressions) and (node and relationship ids). Since Neo4j uses the Cypher (CQL) declarative graph query,

it makes Neo4j vulnerable to injection attacks by using string concatenation. That is because Cypher is vulnerable to injection.

Table 2. Security features of presented NoSQL databases

Security features	NoSQL databases			
	Casandra	MongoDB	REDIS	Neo4J
Auditing	✓	✓	✓	✓
Authentication	✓	✓	✓	✓
Authorization	✓	✓	✓	✓
Communication encryption	✓	✓	✓	✓
Vulnerability to DoS/injection attack	✓	✓	✓	✓

Table 2 shows the security features of the four databases. However, all databases offer security do the data with respect to all categories of the security features presented in this work. Thus, the next section will throw more emphasis on the strengths of each database with regard to each of the security category.

4 Comparative Findings and Discussions

4.1 Security Assessment Key

The following descriptions are key [43] for assessing the security of the databases for all categories;

∗ *High*
A database is considered high with respect to a security category if and only if the features that it provides completely secures the data
∗ *Medium*
If the features needed or provided to secure the data are partial or limited, the database is said to provide medium security with respect to the category.
∗ *Low*
When databases provide no or low required features to secure data.

4.2 Criteria for Assessing Security

With the key established, a description for the categories of security [43] with respect to the metric values are given as;

1. Authentication

- *High* - Logon authentication such as password-oriented, multifactor, certificate, and SSL-based authentication. Logon makes use of a combination of user identifier and password. Examples of logon authentication are captcha images, pin numbers, and biometrics.

  Network-based authentication uses authenticated user session through drivers and network protocol stack.

  IP-based authentication uses IPsec security modules to validate the source and destination IPs.
- *Medium* - The database supports only one means of logon, network-based or IP-based authentication.
- *Low* - No means of authentication or a basic password requirement.

2. Auditing
   - *High* - NoSQL databases must be able to audit and analyze transaction logs(including external and internal activities), database connections and privilege grants.
   - *Medium* - If database can log all user profile activities
   - *Low* - No mechanism to secure the system or data.
3. Authorization
   - *High* - The three levels; database, content, or object level must be supported by the database with some popular models for authorization such as MAC, discretionary, policy-based, task-based, role-based access control(RBAC) and fine-grained access controls.
   - *Medium* - Database must be able to at least support a level of authorization with any of the models under high as well.
   - *Low* - Little or no authorization support by the database.
4. Communication Encryption
   - *High* - NoSQL databases must provide encryption(two broad categories: data-at-rest and data in transit). Examples the former category are MD5 hashing, Data Encryption Standard (DES), AES, SHA1 and SHA2 hashing. Methods of the latter category include SSL, TLS, SSH, and IPsec. Examples of transport level security methods are SSL Record Protocol, Change Cipher Spec Protocol, Secure Shell (SSH), Handshake Protocol, Alert Protocol and IPsec Protocol.
   - *Medium* - Database provides either of the methods of data-at-rest or the methods of transport layer security.
   - *Low* - Database does not provide any encryption method to secure data.
5. Vulnerability to DoS/Injection Attack
   - *High* - Security assurances by the databases include input validation, least privilege policy and secure coding practices.
   - *Medium* - Databases provide only of the mechanisms stated with high security.
   - *Low* - None of the methods are provided by the databases.

Figures 1, 2, 3, 4 and 5 show the results of comparing the four featured NoSQL databases under the categories of security described. In all the figures, on the Y-axis and X-axis are the metric values and NoSQL databases. The metric values

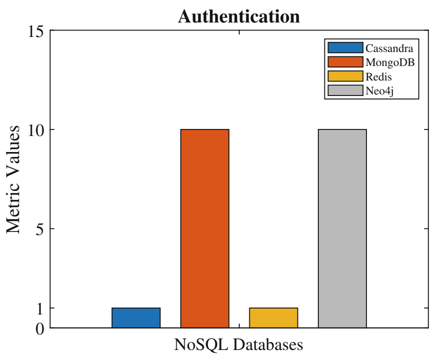

Fig. 1. Comparisons under authentication

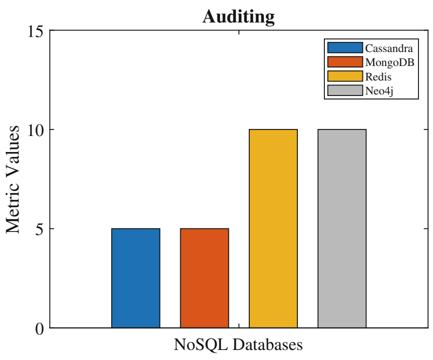

Fig. 2. Comparisons under auditing

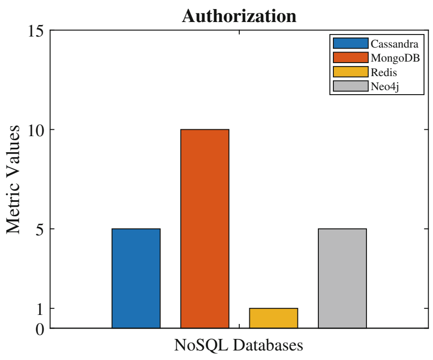

Fig. 3. Comparisons under authorization

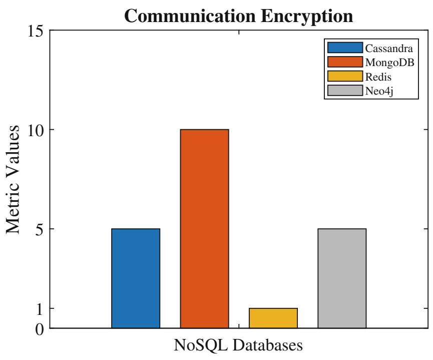

Fig. 4. Comparisons under communication encryption

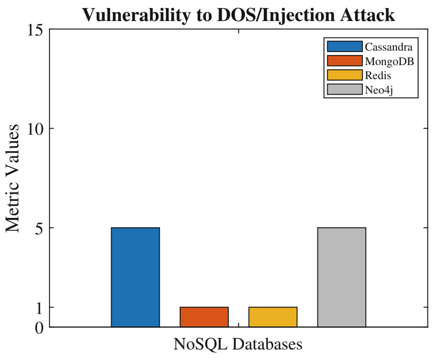

Fig. 5. Comparisons under vulnerability to DoS/injection attack

are labeled 0, 1, 5, 10 and 15, where labels 1, 5 and 10 represent low, medium and high respectively elaborated in Sect. 4.1. In Fig. 1 where a comparison was made under the category "Authentication", databases MongoDB and Neo4j had high security features to secure data. Both Casandra and Redis databases have little or no authentication features at all. Comparisons were made under the category "Auditing" shown in Fig. 2. It can be seen that databases Redis and Neo4j had high features of protecting data under this category while both Casandra and MongoDB have medium or partial protective features. Thirdly, category "Authorization" represented in Fig. 3 has MongoDB with high features, Casandra and Neo4j having medium security features and only Redis having a low protection of data under this category. Figure 4 shows comparisons of NoSQL databases under the security category "Communication Encryption". In this figure only MongoDB has a high security feature. With Casandra and Neo4j providing medium security features, while Redis is either not able to secure data or does it minimally. Finally, presented in Fig. 5 is category "Vulnerability to DoS/Injection Attack". None of the compared NoSQL databases can fully secure data under this category. They either do it partially by Casandra and Neo4j or not at all by databases MongoDB and Redis.

From all five figures, based on combined features and their power to protect data, Neo4j can be said to have the best protective features to secure data. While Redis database performs the weakest in a collective security features in protecting data.

5 Conclusion, Recommendations and Future Works

Given the various security improvements made by NoSQL database platform vendors to improve their security mechanisms, there is still a paucity of research in discussing the security flaws of NoSQL systems as well as the way forward for resolving them. This paper discussed an extensive overview of various vulnerabilities in four of the most common NoSQL databases (MongoDB, Cassandra, Redis and Noe4j) one from each category. In existing works, the comparisons made either included a few NoSQL databases or excluded Neo4j. The algorithms used by each database to support security features were discussed. Each of these databases discussed has its own set of drawbacks and benefits. Comparisons of the databases under different security categories were made as well. Looking at the features identified and comparisons made, NoSQL system developers and administrators can choose and make a better security plan to make their database systems more secure. Despite making significant improvements to improve NoSQL databases, future studies aimed at designing a more robust security framework are required. This should be targeted at designing and implementing a strong security mechanism against the Vulnerability of DoS attacks. Again, further studies can be conducted to design a standard security framework for each category of the NoSQL databases.

References

1. Stonebraker, M., Madden, S., Abadi, D.J., Harizopoulos, S., Hachem, N., Helland, P.: The end of an architectural era: it's time for a complete rewrite. In: Making Databases Work: the Pragmatic Wisdom of Michael Stonebraker, pp. 463–489 (2018)
2. Rocha, L., Vale, F., Cirilo, E., Barbosa, D., Mourão, F.: A framework for migrating relational datasets to NoSQL. Procedia Comput. Sci. **51**, 2593–2602 (2015)
3. Grolinger, K., Higashino, W.A., Tiwari, A., Capretz, M.A.M.: Data management in cloud environments: NoSQL and NewSQL data stores. J. Cloud Comput. **2**(1), 1–24 (2013). https://doi.org/10.1186/2192-113X-2-22
4. Han, J., Haihong, E., Le, G., Du, J.: Survey on NoSQL database. In: 2011 6th International Conference on Pervasive Computing and Applications, pp. 363–366. IEEE (2011)
5. Lakshman, A., Malik, P.: Cassandra: a decentralized structured storage system. ACM SIGOPS Oper. Syst. Rev. **44**(2), 35–40 (2010)
6. DeCandia, G., et al.: Dynamo: Amazon's highly available key-value store. ACM SIGOPS Oper. Syst. Rev. **41**(6), 205–220 (2007)
7. Chang, F., et al.: Bigtable: a distributed storage system for structured data. ACM Trans. Comput. Syst. **26**(2), 1–26 (2008)
8. Kalid, S., Syed, A., Mohammad, A., Halgamuge, M.N.: Big-data NoSQL databases: a comparison and analysis of "Big-Table", "DynamoDB", and "Cassandra". In: 2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA), pp. 89–93. IEEE (2017)
9. Chebotko, A., Kashlev, A., Lu, S.: A big data modeling methodology for Apache Cassandra. In: 2015 IEEE International Congress on Big Data, pp. 238–245. IEEE (2015)
10. Rosselli, M., Niemann, R., Ivanov, T., Tolle, K., Zicari, R.V.: benchmarking the availability and fault tolerance of Cassandra. In: Rabl, T., Nambiar, R., Baru, C., Bhandarkar, M., Poess, M., Pyne, S. (eds.) WBDB -2015. LNCS, vol. 10044, pp. 87–95. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49748-8_5
11. Jogi, V.D., Sinha, A.: Performance evaluation of MySQL, Cassandra and HBase for heavy write operation. In: 2016 3rd International Conference on Recent Advances in Information Technology (RAIT), pp. 586–590. IEEE (2016)
12. Abramova, V., Bernardino, J.: NoSQL databases: MongoDB vs cassandra. In: Proceedings of the International C* Conference on Computer Science and Software Engineering, pp. 14–22 (2013)
13. Why use MongoDB and when to use it? MongoDB. https://www.mongodb.com/why-use-mongodb
14. Kookarinrat, P., Temtanapat, Y.: Analysis of range-based key properties for sharded cluster of MongoDB. In: 2015 2nd International Conference on Information Science and Security (ICISS), pp. 1–4. IEEE (2015)
15. Arora, R., Aggarwal, R.R.: Modeling and querying data in MongoDB. Int. J. Sci. Eng. Res. **4**(7), 141–144 (2013)
16. Indexes. https://www.mongodb.com/docs/manual/indexes/
17. Top 5 features of MongoDB: MongoDB. https://www.mongodb.com/what-is-mongodb/features
18. Dipina Damodaran, B., Salim, S., Vargese, S.M.: Performance evaluation of MySQL and MongoDB databases. Int. J. Cybern. Inform. (IJCI) **5** (2016)

19. Bugiotti, F., Cabibbo, L.: A comparison of data models and APIs of NoSQL datastores. In: SEBD, pp. 63–74 (2013)
20. Redis: in-memory data store. How it works and why you should use it. https://aws.amazon.com/redis/
21. Ji, Z., Ganchev, I., O'Droma, M., Ding, T.: A distributed Redis framework for use in the UCWW. In: 2014 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, pp. 241–244. IEEE (2014)
22. Chen, S., Tang, X., Wang, H., Zhao, H., Guo, M.: Towards scalable and reliable in-memory storage system: a case study with Redis. In: 2016 IEEE Trustcom/BigDataSE/ISPA, pp. 1660–1667. IEEE (2016)
23. Li, S., Jiang, H., Shi, M.: Redis-based web server cluster session maintaining technology. In: 2017 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), pp. 3065–3069. IEEE (2017)
24. Redis replication. https://redis.io/docs/manual/replication/
25. Pan, C., Wang, X., Luo, Y., Wang, Z.: Penalty-and locality-aware memory allocation in Redis using enhanced AET. ACM Trans. Storage **17**(2), 1–45 (2021)
26. Redis data structures. https://redis.com/redis-enterprise/data-structures/
27. Jana, T.: Achieve unrivaled speed and scalability with Neo4j. https://neo4j.com/blog/achieve-unrivaled-speed-and-scalability-neo4j/
28. Graph modeling guidelines. https://neo4j.com/developer/guide-data-modeling/
29. Holzschuher, F., Peinl, R.: Performance of graph query languages: comparison of cypher, gremlin and native access in Neo4j. In: Proceedings of the Joint EDBT/ICDT 2013 Workshops, pp. 195–204 (2013)
30. Indexes for search performance. https://neo4j.com/docs/cypher-manual/current/indexes-for-search-performance/
31. Kapadia Gayatri, S., Morena Rustom, D.: Comparative study of role based access control in cloud databases and NoSQL databases. Int. J. Adv. Res. Comput. Sci. **8**(5), 51–57 (2017)
32. Savaram, R.: Apache Cassandra data security management. https://mindmajix.com/cassandra/data-security-management
33. Audit Logging: Apache Cassandra documentation. https://cassandra.apache.org/doc/latest/cassandra/new/auditlogging.html
34. Adia: CVE-2021-44521: RCE vulnerability in Apache Cassandra. https://jfrog.com/blog/cve-2021-44521-exploiting-apache-cassandra-user-defined-functions-for-remote-code-execution/
35. Authentication - MongoDB manual. https://www.mongodb.com/docs/manual/core/authentication/
36. Okman, L., Gal-Oz, N., Gonen, Y., Gudes, E., Abramov, J.: Security issues in NoSQL databases. In: 2011 IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications, pp. 541–547. IEEE (2011)
37. Sahafizadeh, E., Nematbakhsh, M.A.: A survey on security issues in Big Data and NoSQL. Adv. Comput. Sci. **4**(4), 68–72 (2015)
38. Auditing - MongoDB manual. https://www.mongodb.com/docs/v4.4/core/auditing/
39. Haber, I.: Using stunnel to secure. https://redis.com/blog/stunnel-secure-redis-ssl/
40. Authentication and authorization - upgrade and migration guide. https://neo4j.com/docs/upgrade-migration-guide/current/migration/surface-changes/auth/
41. Sasaki, B.M.: Neo4j data encryption with OGM [Community Post]. https://neo4j.com/blog/neo4j-data-encryption-ogm/

42. SSL framework - operations manual. https://neo4j.com/docs/operations-manual/current/security/ssl-framework/
43. Zahid, A., Masood, R., Shibli, M.A.: Security of sharded NoSQL databases: a comparative analysis. In: 2014 Conference on Information Assurance and Cyber Security (CIACS), pp. 1–8. IEEE (2014)
44. Nehra, M.: Top 10 NoSQL databases in 2022 (2022). https://www.decipherzone.com/blog-detail/nosql-databases. Accessed 14 Aug 2022
45. Li, Z.: NoSQL databases (2019)