5-2020

# Assessing the vulnerabilities and securing MongoDB and Cassandra databases

sindhu gadhiraju
gadhirajusindhura@gmail.com

**Assessing the vulnerabilities and securing MongoDB and Cassandra databases**

by

Sindhu Gadhiraju

A starred paper

Submitted to the Graduate Faculty of

St. Cloud State University

In Partial Fulfillment of Requirements

for the Degree of

Master of Science

In Information Assurance

May 2020

Committee members:
Abdullah Abu Hussein
Mark B. Schmidt
Balasubramanian Kasi

## Abstract

Due to the increasing amounts and the different kinds of data that need to be stored in the databases, companies, and organizations are rapidly adopting NoSQL databases to compete. These databases were not designed with security as a priority. NoSQL open-source software was primarily developed to handle unstructured data for the purpose of business intelligence and decision support. Over the years, security features have been added to these databases, but they are not as robust as they should be, and there is a scope for improvement as the sophistication of the hackers has been increasing. Moreover, the schema-less design of these databases makes it more difficult to implement traditional RDBMS like security features in these databases. Two popular NoSQL databases are MongoDB and Apache Cassandra. Although there is a lot of research related to security vulnerabilities and suggestions to improve the security of NoSQL databases, this research focusses specifically on MongoDB and Cassandra databases. This study aims to identify and analyze all the security vulnerabilities that MongoDB and Cassandra databases have that are specific to them and come up with a step-by-step guide that can help organizations to secure their data stored in these databases. This is very important because the design and vulnerabilities of each NoSQL database are different from one another and hence require security recommendations that are specific to them.

# Table of Contents

# List of Tables

Table Page

## List of Figures

# Chapter I: Introduction

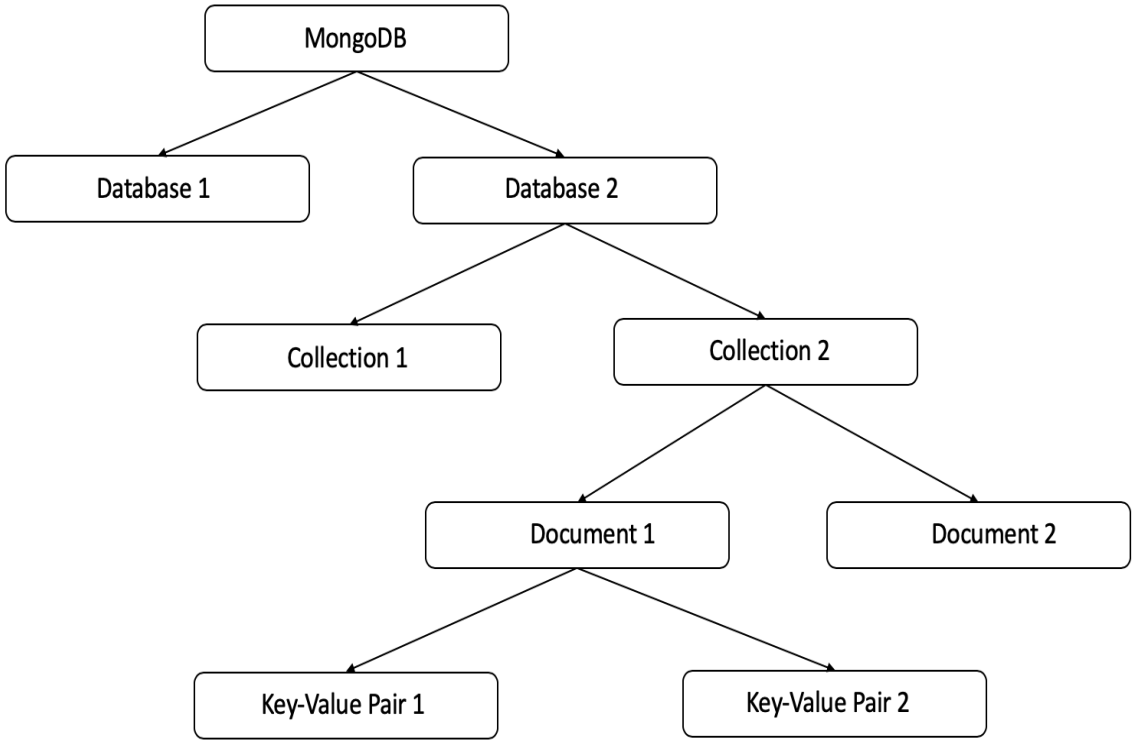
## Introduction

NoSQL (i.e., non-SQL or not only SQL) database is a database design, which is not based on SQL (Structure query language). Essentially, NoSQL databases are not relational. They are designed with looser consistency models than RDBMS and usually do not have a schema. NoSQL databases do not rely on schemas, tables, rows, or columns to organize and retrieve data. NoSQL databases are particularly useful to store semi-structured and unstructured data. NoSQL is being adopted by companies as a complement to RDBMS to address new use cases because of the increasing volumes, speed, variety (semi-structured and unstructured) of information that companies need to store on a daily basis and the need for frequent updates and features as the business requirements change in this digital and competitive economy is becoming more and more difficult with the existing traditional database management tools. Some examples of NoSQL databases are MongoDB, Cassandra, CouchDB, Redis, and HBase. Two of the top-rated NoSQL databases are MongoDB and Cassandra (Cooke, 2018).

MongoDB is a document-oriented database, which means the data is stored in the form of a document. Each database consists of collections, which in turn consist of documents. Each document consists of different number of fields, size, and content. Each document has an ID field, which is used as a primary key (Saran, Sai Baba, Jayanthi & Soundararanjan, 2015). The structure of MongoDB is shown in the below figure. MongoDB does not have to have a schema that is defined beforehand. The records (i.e., fields) can be created on the fly.  MongoDB has its own query language, which is called Mongo query language.

**Figure 1**

*Structure of MongoDB*




*Note.* A representation of the structure of MongoDB (Saran et al., 2015)

The documents in MongoDB are JSON like, i.e., documents are represented in a binary-encoded format, which is called BSON (Binary JSON). BSON is an extension of the JSON model to provide ordered fields and additional data types.

The default configuration of MongoDB allows full access of the database to anyone. MongoDB databases have a history of theft, and MongoDB servers have been held for a ransom

(McCallion, 2017). Since December of 2016, ransomware attacks have been happening on MongoDB databases, where attackers wipe off the database and ask for a ransom to get the data back. In 2018, the California Voter database, which contained information of over 19 million voters in California, was exposed online due to an unsecured MongoDB database (Cimpanu, 2018). The database contained personal information like names, contact information, addresses, registrant ID, etc. An attacker used an automated script that scanned the internet for open MongoDB database and deleted its content and left a ransom note behind asking for 0.2 bitcoin as payment to get the data. It is unclear who owned the database, but it is suspected that it could be the state government, a contractor, or another hacker who stole the database from the state's real database.

Apache Cassandra is a wide-column store database. It was developed at Facebook originally for inbox search. It was designed to manage and handle large amounts of data across many servers. It can very quickly ingest as well as process very large amounts of data. It is a distributed, decentralized, highly scalable, available tuneably consistent, and fault-tolerant database. It has identical nodes that are clustered together in order to eliminate bottlenecks and single points of failure. Cassandra uses a peer-to-peer distribution model in order to distribute data. All nodes in Cassandra play an identical role, communicating with each other equally unlike the master slave model. Cassandra database is being used by some of the biggest companies such as Twitter, Cisco, eBay, Facebook, Netflix etc. CQL (Cassandra query language) is used to query the Cassandra database. Although Cassandra has much better security than most NoSQL databases, there are some vulnerabilities that can be exploited. For example,

the default configuration of some versions is vulnerable to remote code execution. The security of these databases needs to be improved because some organizations also store sensitive information in these databases.

**Definition of Terms**

- **NoSQL:** Non-SQL /non-relational/not only SQL is a schema-less database where data is stored in forms other than tabular relations unlike relational databases.
- **SQL:** Structured query language is a programming language used to manage data in the relational database management system (RDBMS)
- **RDBMS:** Relational database management system (RDBMS) is a database management system that is based on the relational model.
- **Confidentiality:** Confidentiality means the data is accessible only to the people who are authorized to access it based on a set of rules. It limits access to the data.
- **Integrity:** Integrity means making sure that the data is consistent, accurate, and trustworthy over its entire life cycle.
- **Availability:** Availability ensures that authorized people have reliable access to the data at all times.
- **JavaScript:** JavaScript is a high level, multi-paradigm, and interpreted programming language, which is an essential part of web applications as it enables interactive web pages.
- **PHP:** Personal home page is a scripting language on the server-side and used for web development.

- **JSON:** JavaScript object notation is a file format derived from JavaScript and is language independent. It transmits data objects that contain attribute-value pairs and array data types. It transmits them in human-readable text.
- **NoSQL injection:** NoSQL injection is a vulnerability in the NoSQL database that allows an attacker to control the database queries with the help of unsafe user input. It can be used to modify data, change privileges, expose sensitive information, or take down the entire application.
- **CSRF:** Cross-site request forgery is a type of cyber-attack that tricks an end-user into executing malicious actions on a web application that they are authenticated in.
- **DOS:** Denial of service attack is a type of cyber-attack in which an attacker makes the system or network resource unavailable to the intended users by disrupting services either temporarily or indefinitely.
- **MD5:** Message digest algorithm is a popular hashing function that produces a 128-bit hash value. It is mainly used to prove the integrity of the data.
- **RESTful API:** RESTful API is an API (Application program interface) that uses HTTP requests in order to PUT, GET, POST and DELETE data.
- **Password brute force attacks:** A password brute force attack or brute force attack is an attack where the attacker submits a list of passwords and checks all of them systematically to find the correct one.
- **Man in the middle attacks:** A man in the middle attack is a kind of cyber-attack where an attacker relays and alters the communication between two parties secretly while they believe they are communicating with each other.

- **Authorization:** Authorization is a process that determines what permissions a user has, which determines what a user can see and do.
- **Authentication:** Authentication is a process that confirms a user's identity, mainly with the help of usernames and passwords so that only authorized users have access to a system, resource, or data.
- **Encryption:** Encryption is a process of encoding the message or data in such a way that only authorized people can access it using a predetermined key.
- **TLS/SSL:** Transport layer security and Secure Sockets layer are cryptographic protocols that provide security for communications over a computer network (SSL is a deprecated predecessor of TLS).
- **Plain-text:** Plain text / Clear text is data that is readable without any encryption or graphical representation.
- **POST:** POST is an HTTP supported request method used by the WWW (World Wide Web). The request is for the webserver to accept data that is enclosed in the body of the request message. It is mainly used when submitting a completed web form or uploading a file.
- **HTML:** Hypertext markup language is the standard markup language that is used to create web pages and web applications.
- **Cluster:** In databases, clusters are a collection of databases that connect together to provide a service.

## Problem Statement

Security of NoSQL databases is weak when compared to the relational database management systems as security was not a priority while designing these databases. NoSQL databases are vulnerable to various attacks and are weak in fine-grained authentication and access controls due to the lack of a structure or schema.

With the increasing attacks and sophistication of hackers, NoSQL databases need better security enhancements to protect the sensitive information stored in them. With many companies adopting NoSQL databases to meet availability, better performance, and scalability, it has to be made sure that the security of NoSQL databases is at least comparable to traditional relational database management systems (RDBMS) if not more.

There seems to be a lack of comprehensive studies of security risks and vulnerabilities and security recommendations that are associated with the most recent versions of MongoDB and Cassandra.

## Nature and Significance of the Problem

Securing a relational database is different from securing a NoSQL database. The security features that are used in relational databases like access control systems, integrity, and encrypted communication are difficult to be implemented in the NoSQL database because of their design. This research is important because there seems to be a lack of a step by step guide for securing MongoDB and Cassandra and this research would benefit small companies and organizations that are planning to use NoSQL databases to store semi-structured and unstructured data.

**Objective of the Study**

The objective of the study is to identify and analyze all the security vulnerabilities that MongoDB and Cassandra databases have that are specific to them and come up with a step by step guide for each database that can help organizations to secure their data stored in these databases.

**Study Questions**

The following are the study questions for this research:

1. What are the security vulnerabilities of NoSQL databases?
2. What are the security vulnerabilities of MongoDB and Cassandra NoSQL databases?
3. What are the current security features that MongoDB and Cassandra provide?
4. Are these security features enough to guarantee the security of these databases, and if not, how can these be improved?
5. What can companies that use MongoDB and Cassandra databases do to secure their data in these databases?

**Summary**

In this chapter, a brief introduction for NoSQL databases, MongoDB database, and Apache Cassandra database has been given along with an example of a ransomware attack on a MongoDB database. This chapter explains

the problem, why the research is important, and the objectives of the research.

# Chapter II: Background and Review of Literature

## Introduction

The schema of NoSQL databases is dynamic, unlike SQL databases that have a pre-defined schema. NoSQL databases provide high scalability, performance, low latency, and flexibility. These databases use unstructured query language, whose syntax differs from database to database.

NoSQL databases are classified based on how they store data. There are basically five different types of NoSQL databases based on how data is stored (Vishwakarma, 2017). They are:

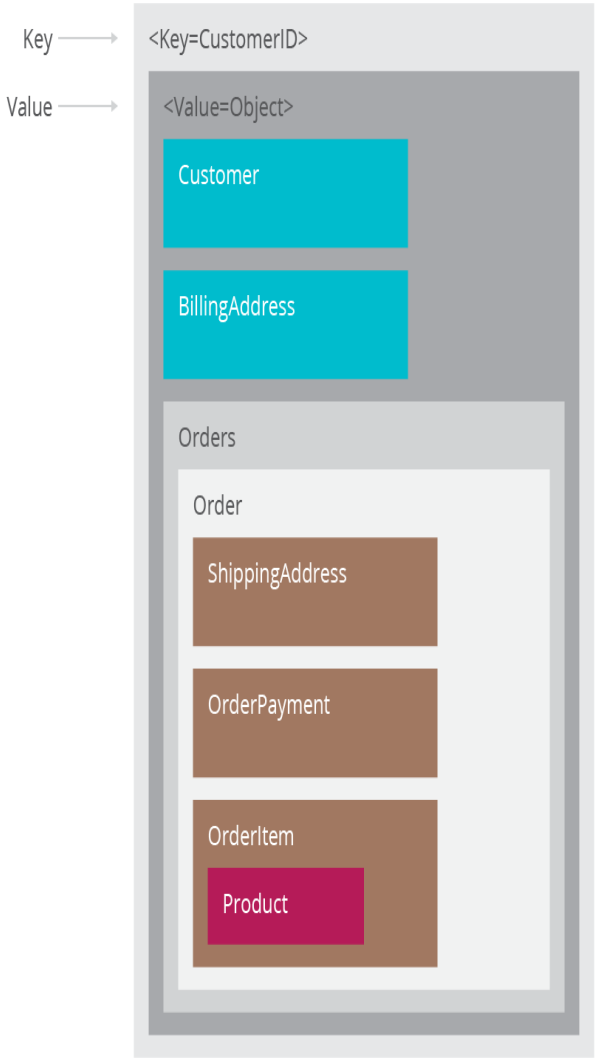
(i) Key-Value Store: These are used to store keys and their associated paired values. Because of its simplicity, the querying is fast. Some use cases of key-value databases are: To store user session data, to store user preferences, to store shopping cart data, and to maintain schema-less user profiles.

Some popular Key-Value based NoSQL databases are Dynamo and Riak. Some of the popular companies that use Key Value-based NoSQL databases are Twitter, Coinbase and Pinterest.

The diagram below shows a key-value database for customer orders. The CustomerID is the key, and the value stores the customer name, billing address, and order details like shipping address, order payment, and order item.

**Figure 2**

*Key value database*




*Note*. An example of key-value database (Sadalage, 2014)

(ii) Document-based store: These are similar to key-value store databases, but instead of values, these store documents associated with the keys. The type of documents stored are JSON, XML, or BSON. In these databases, the documents are stored in the value part of the key-value store database. Some use cases of document store databases are blogging platforms, analytics platforms, content management systems, and E-commerce platforms. Some popular document-based NoSQL databases are CouchDB and MongoDB. Some of the popular companies that use Document-based NoSQL databases are Cisco and SEGA. The below shows an example of a document written in JSON. In the document store database, this document can be retrieved by referring to the ‘customerid’.

**Figure 3**

*Structure of MongoDB*

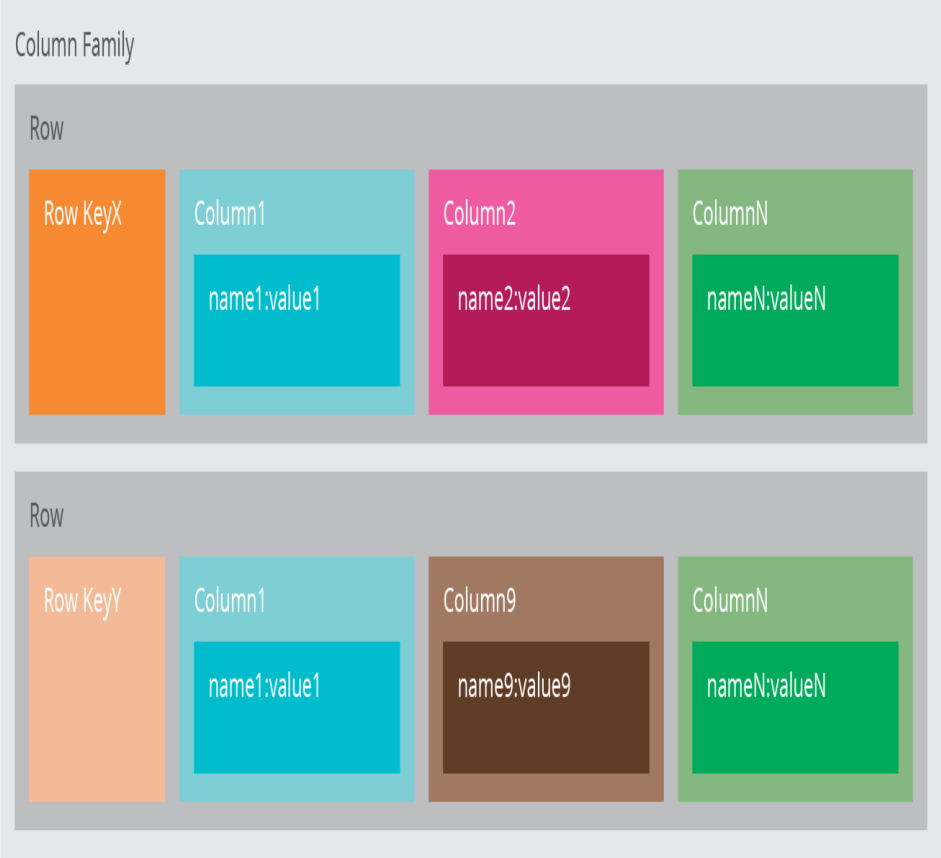

*Note*. An example of Document database (Sadalage, 2014)

(iii) Column-based store: In Column based NoSQL database, the data is stored in columns instead of rows. Each column is associated with a column key. Unlike RDBMS, which reads and writes rows of data, a column-based store is designed for reading and writing columns of data. Some use cases of column databases are blogging platforms, content management systems, and systems that maintain counters.

Some popular column-based NoSQL databases are Cassandra and HBase. Some of the popular companies that use Column based NoSQL databases are Facebook and Spotify. The below diagram shows an example of a column store database. The columns in each row here are contained within that particular row, and each row can have a different number of columns, and they can be in a different order and data types, etc.

**Figure 4**

*Column bases database*



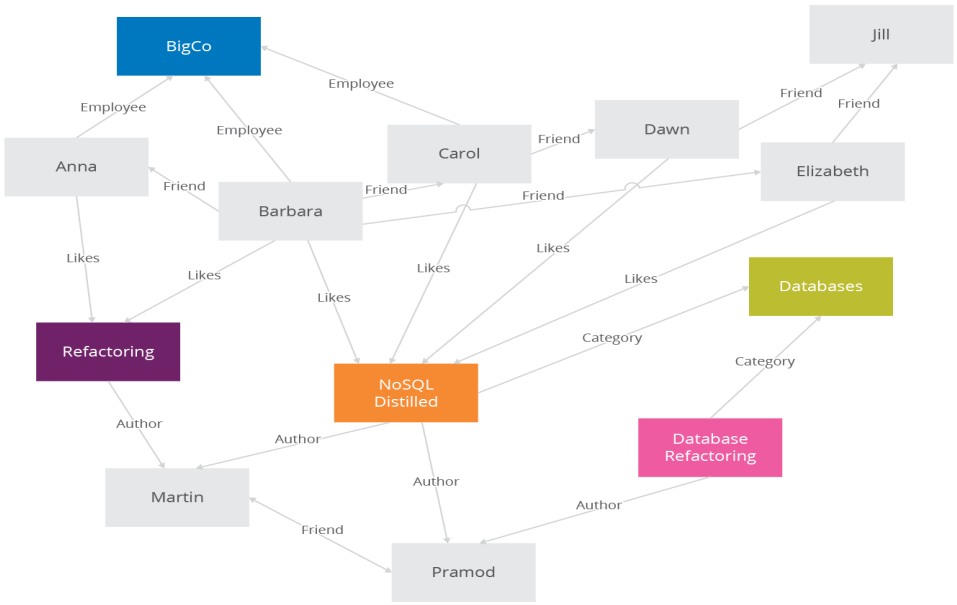

*Note.* An example of Column based database (Sadalage, 2014)

(iv) Graph-based: These are used to store information about interconnected data like networks. This database is based on nodes and relationships. Some use cases of graph-based NoSQL databases are Social networks, graph-based search, fraud detection, and network and IT operations.

Some popular graph-based NoSQL databases are Neo4J and Allergo. Some of the popular companies that use Graph-based NoSQL databases are Walmart and Cisco. The below diagram shows an example of a graph-based NoSQL database. The rectangles are nodes and contain data. The arrows represent the relationships between the nodes.

**Figure 5**

*Graph database*



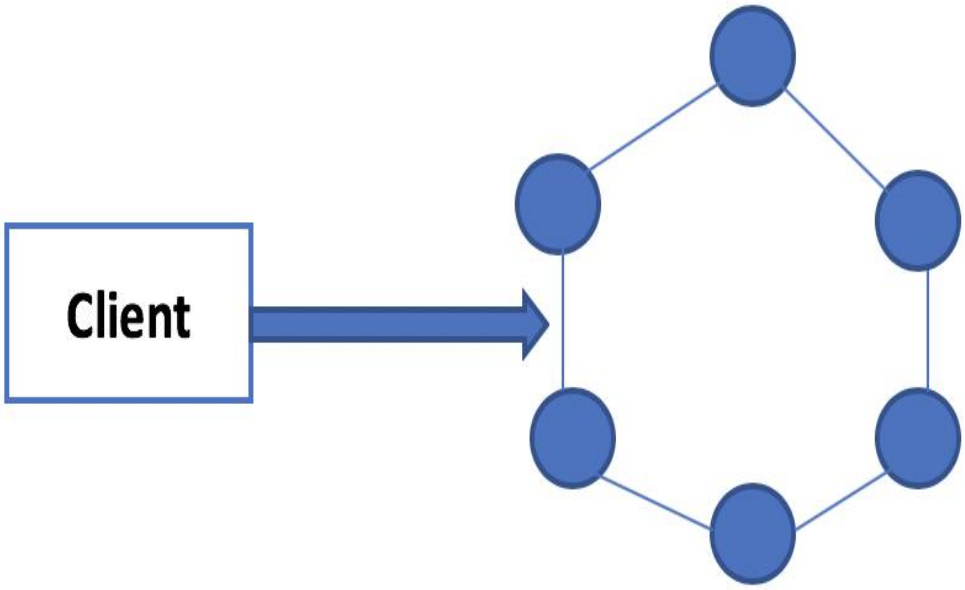

Note. An example of Graph database (Sadalage, 2014)

MongoDB is a schema-free, distributed, and highly available and scalable NoSQL document-based database (Dayley, 2014). It contains one or more collections of JSON style documents (BSON). A document is a collection of fields. MongoDB is open-source, and its scripting was done in C++ programming language. It can access big data at very high speeds. It can handle complex data types. MongoDB has its own query language called Mongo query language. It is used by some of the big companies like Craigslist, MTV Networks and The New York Times.

Apache Cassandra is one of the leading NoSQL distributed database management system that can manage large amounts of data across many commodity servers (Fedak, 2018). It can provide high scalability, flexibility, performance, and availability. Cassandra uses its own query language called Cassandra query language (CQL). Apache Cassandra has multiple nodes that play an identical role, unlike in a master-slave model, as shown in the below figure. Big companies like Apple, Spotify, Uber, and Netflix are using Apache Cassandra.

**Figure 6**

*Cassandra Nodes*




*Note.* A representation of Cassandra nodes

**Literature Related to the Problem**

(Dadapeer, Indravasan & Adarsh, 2016) in " A survey on security of NoSQL Databases " have discussed the issues in the NoSQL database security in general and the security issues specific to popular NoSQL databases like Cassandra, MongoDB, Redis, CouchDB, and HBase. They have also provided suggestions on mitigating two major types of attacks on NoSQL databases, i.e., injection attack and the REST API exposure and CSRF attacks.

(Dindoliwala & Morena, 2017) in " Survey on security mechanisms in NoSQL databases" have explained the various data storage models for NoSQL and discussed the security features that are provided by Cassandra, MongoDB, Gemstone, db4o, and Objectivity/DB. They have concentrated mainly on data encryption, authentication, authorization, and auditing features. They have briefly discussed the security challenges in NoSQL databases. According to the paper, because the NoSQL databases are schema-free, fine-grained access control and role-based access control are difficult to implement. Also, there is no security feature for embedding security within the database. They have suggested embedding security in the middleware.

(Chahal, Kharb & Gupta, 2017) in " Challenges and Security issues in NoSQL databases" have discussed data in rest, data in motion, data in use, various types of authentication, different levels of authorizations, and data encryption, which they suggest focusing on before selecting a NoSQL database for the organization. They have also discussed major NoSQL database vulnerabilities such as connection pooling, key Brute-forcing, HTTP REST API, and the Denial of Services (DOS) attack. They have also briefly described the problem with MongoDB older version not designed to bind itself from the localhost, which leads to data leakage. The paper states that MongoDB and Cassandra databases lack file encryption, have weak authentication and simple authorization, and are vulnerable to injections and DOS attacks.

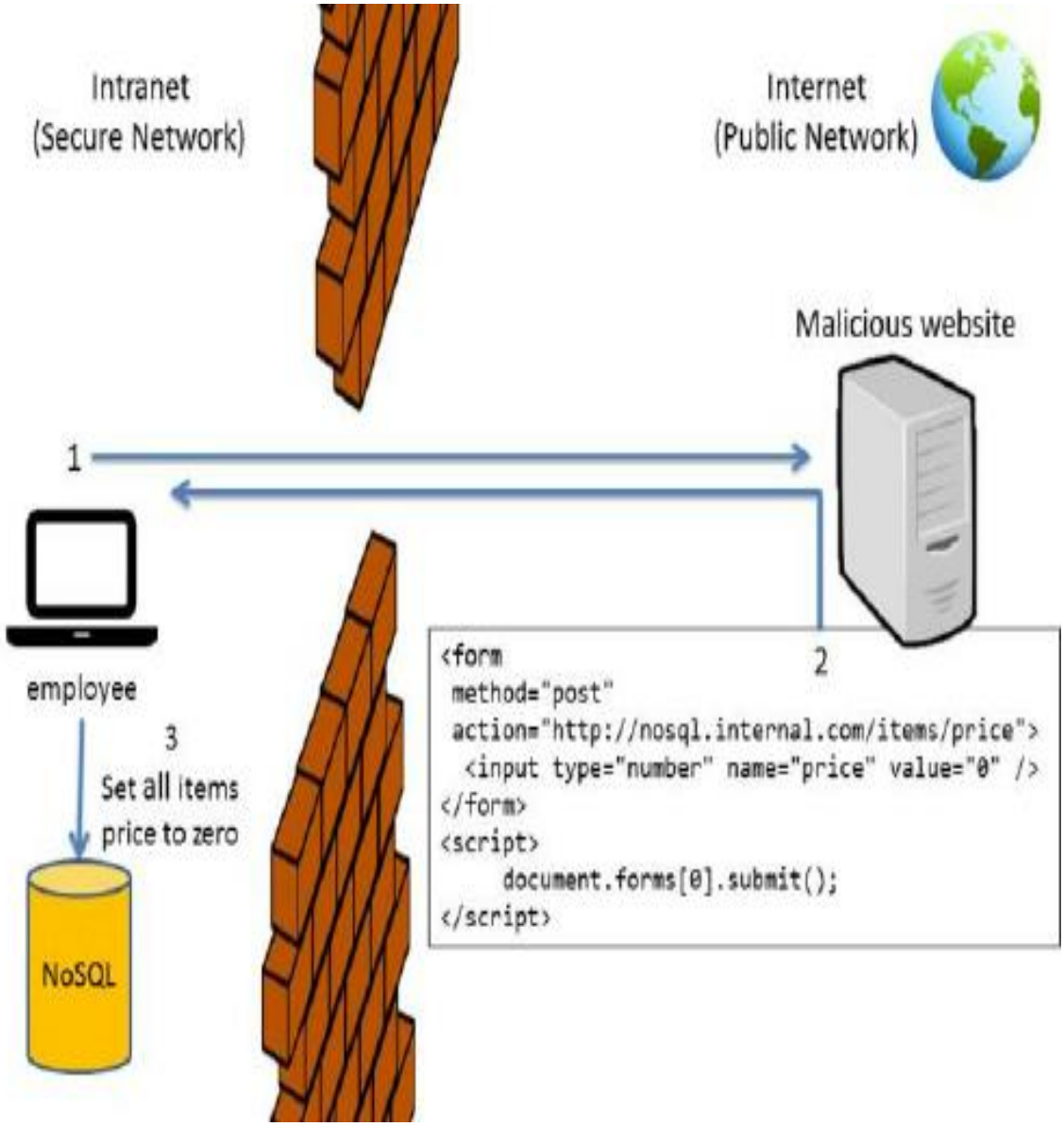
(Okman, Gal-Oz, Gonen, Gudes, & Abramov, 2011) in "Security issues in NoSQL databases" have discussed two most popular NoSQL databases: MongoDB and Cassandra and given an overview of their security features and security issues. They have also provided their

recommendations for how to mitigate the security issues that arise from the vulnerabilities in these databases. They have concluded that the common problems that both databases have are a lack of data files encryption, weak authentication, simple authorization, lack of support for RBAC, and susceptible to injections and denial of service attacks.

(Aviv, Shulman-Peleg & Bronshtein, 2015) in “No SQL, No injection? Examining NoSQL Security” have demonstrated in the paper how the JSON format, which is used to represent the queries and data in MongoDB database, allows for new types of injection attacks. They have discussed how PHP array injections can allow a hacker to log into an application without any authentication. They have also demonstrated Javascript encryption and explained how the exposure of  HTTP REST API, which is a common feature of NoSQL databases to query the database from client applications, could expose the database to CSRF attacks, as shown in the below figure. They have also recommended some injection mitigation techniques like running dynamic application security testing (DAST) and static code analysis to find injection vulnerabilities in the code, controlling requests, and limiting the format to protect against risks from API exposure and using access control and authentication.

**Figure 7**

*CSRF via NoSQL REST API*




*Note.* A depiction of how CSRF can be performed via NoSQL REST API (Aviv et al., 2015)

(Noiumkar & Chomsiri, 2014) in “A comparison the level of security on Top 5 open source NoSQL databases” have used five security factors: Data file encryption, Client/Server Authentication/Encryption, Inter-cluster Authentication/Encryption, Script injection and denial of service attacks to evaluate and compare the security of top five open-source NoSQL databases which are: MongoDB, Cassandra, CouchDB, Hypertable and Redis. According to the paper, all five databases do not have data file encryption, MongoDB and CouchDB are vulnerable to script injection, Cassandra and CouchDB are vulnerable to denial of service attacks and CouchDB is the only one having a good client/server authentication/encryption and inter-cluster authentication/encryption. The researchers have given some recommendations on encrypting data in application level and using a tunnel to provide safer communication for servers in order to make these databases more secure.

(Zahid, Masood & Shibli, 2014) in “Security of Sharded NoSQL databases: A comparative analysis” have discussed the assessment criteria for evaluating the security of sharded NoSQL databases. The criteria that they used are based on authentication, access controls, secure configurations, data encryption, and auditing. Based on these criteria, they have analyzed six different NoSQL sharded databases in terms of security features. They have done that by using three metric values for each factor of security criteria as Low, Medium, and High. They have illustrated comparative results for the six different NoSQL sharded databases for each factor of security criteria using the metrics with the help of bar graphs.

(Hou, Qian, Li, Shi, Tao & Liu, 2016) in “MongoDB NoSQL injection analysis and detection” have demonstrated experimental testing of NoSQL injections on a MongoDB

database using JavaScript and PHP to examine its security. To demonstrate the attack, they have used the example of a library system that uses MongoDB database to store all books related information. They demonstrated NoSQL injections on the database in two ways. One is by using the input boxes to inject, and the other is injecting by URL. They have also suggested two methods to mitigate these types of injections in code level. They suggested that developers add a JavaScript code in order to limit the input boxes in the system/software building state. They suggested using a parameterized statement to check and filter the variables. They also suggested using security layers, such as a malicious feature detection system that can detect whether the system/software has any features that are not secure.

(Shahriar & Haddad, 2017) in "Security vulnerabilities of NoSQL and SQL databases for MOOC Applications" have compared traditional SQL databases and NoSQL databases and discussed the vulnerabilities that are inherent to the two most popular NoSQL databases that are: MongoDB and Cassandra. MOOC (Massive open online courses) provide free or inexpensive access to online educational courses for learners. As these courses are deployed on open source database management systems, which are shifting rapidly towards NoSQL databases due to an increase in data generated, the authors want to increase awareness of threats that arise when interacting online with platforms that deploy NoSQL databases. The authors have compared SQL and NoSQL databases in terms of the data model, schema, normalization, scalability, data manipulation, and integrity. They have given an overview of MongoDB and Cassandra and summarized the issues in NoSQL databases as encryption, inter-node communications, authentication, authorization, audit, and data consistency. They have discussed these issues for

MongoDB and Cassandra as well. They have also given examples of NoSQL injection, DoS, and XSS attacks on MongoDB and CQL injection, DoS, and XSS attacks on Cassandra. They have also discussed the vulnerabilities and attacks on the MySQL database. They have suggested creating standards and implementing encryption to protect NoSQL databases.

From the above literature, the vulnerabilities and issues can be summarized as:

**Security issues or vulnerabilities in NoSQL databases in general:**

(i) As the NoSQL database is schema free, it is very difficult to implement fine-grained access control or enforce role-based access control. This, combined with a lack of central control, makes it very difficult to enforce integrity constraints (Dindoliwala & Morena, 2017).

(ii) Security has to be imposed in the middleware by the developers for NoSQL databases as there is no feature to embed security within the database (Dindoliwala & Morena, 2017).

(iii) NoSQL databases have distributed nodes, which creates an increased attack surface that makes it difficult to secure these databases. If one node is compromised, the entire system can be compromised. In a NoSQL database, data is shared between thousands of nodes. This means there would be multiple entry points associated with each node, which increases the possibility of unauthorized access (Kadebu, Prudence, & Mapana, 2014).

(iv) NoSQL databases that use JavaScript and PHP on the server-side for the purpose of enhancing database performance are vulnerable to query injection attacks (Shahriar & Haddad, 2017).

(v) NoSQL databases have very fewer security measures within the database when compared to traditional SQL databases. For example, traditional SQL databases have built-in data integrity and encryption features, whereas NoSQL databases store data in plain text and lack in such inbuilt security features. External security mechanisms must be implemented to secure these databases (Dindoliwala & Morena, 2017).

(vi) NoSQL databases are prone to password brute force attacks, replay attacks, and man in the middle attacks due to inefficient password storage mechanisms and authentication techniques (Dindoliwala & Morena, 2017).

(vii) NoSQL databases mainly use REST as their communication protocol, which is prone to injection attacks, cross-site request forgery, and cross-site scripting attacks (Dindoliwala & Morena, 2017).

(viii) NoSQL databases lack authentication mechanisms that can be enforced across all the nodes of the cluster. The current authentication mechanisms work on a local node level (Shahriar & Haddad, 2017).

(ix) In NoSQL databases, authorization is applied on a per-database level and not on a collection level. Also, authorization is applied at higher levels rather than lower levels (Dadapeer et al., 2016).

(x) As NoSQL databases may contain sensitive data, the inefficient security mechanisms make the database vulnerable to insider attacks as well. This is made more problematic by the fact that most NoSQL databases lack good logging, auditing and log analysis mechanisms (Dadapeer et al., 2016).

(xi) A lot of NoSQL databases lack network transport layer encryption over the TLS/SSL on both server and client- side. This leads to insecure communication between the server and the clients (Shahriar & Haddad, 2017).

(xii) In key-value NoSQL databases, it is very important to protect the key. As NoSQL databases are schema-free, there is no need to find the schema, and this makes it easy for an attacker to find or decrypt the key using key brute forcing attack (Chahal et al., 2017).

**Security issues or vulnerabilities in MongoDB:**

(i) MongoDB does not have the facility to automatically encrypt files that are written to the database. They are stored in plain-text. This means that if a hacker is able to get into the system, he or she can easily read these files (Dadapeer et al., 2016).

(ii) Authentication can be enabled in standalone mode, but when using sharded mode in MongoDB, authentication is not supported. The authentication provided in standalone mode uses a key that is hashed in MD5 before it is stored in the key file. This is relatively secure, but if the attacker cracks the MD5, he/she can crack the key if they get a hold of the key file. The enterprise edition for MongoDB does provide an additional service for Kerberos, but the authentication is not supported in sharded mode (Noiumkar & Chomsiri, 2014).

(iii) By default, in MongoDB, authorization is disabled. The authorization is provided on a per-database level, and it follows a role-based approach. Also, the roles are limited to a few (Shahriar & Haddad, 2017).

(iv) The internal scripting language used in MongoDB is JavaScript, which is not a secure scripting language and is vulnerable to a scripting injection attack (Aviv et al., 2015).

(v) MongoDB uses JSON format for data and queries. Although JSON format is considered more secure than SQL in terms of conducting an injection attack, it does allow for new types of injection attacks. It is vulnerable to PHP array injections, JavaScript injection, and cross-site request forgery attacks due to exposure of HTTP REST API (Aviv et al., 2015).

(vi) MongoDB has a feature of exposing HTTP REST API, which lets the client applications to query the database. This feature makes the database vulnerable to CSRF attacks that allow bypassing firewalls and other external perimeter defenses by the attacker. In a secured network, when the database exposes REST API, anyone with access to the network can query the database using HTTP, which lets queries to be initiated from the browser. This is a huge vulnerability as an attacker can inject a website with an HTML form, and spear fishing can be used by an attacker controlling a malicious website to trick an employee of a company into browsing on that website. If the employee does that, the HTML form can be submitted with an action URL of an internal MongoDB database. The action will succeed as the employee has access to the network from within (Aviv et al., 2015).

(vii) MongoDB does not have any facilities for auditing actions that are performed in the database. For each instance of MongoDB, there is, however, an HTTP console that displays information about the system and the clients that connect. This would be of no use though if authorization is disabled (Shahriar & Haddad, 2017).

(viii) MongoDB's internal HTTP server does not support SSL for client node communication, which means the client communications are not secure unless the enterprise edition is used or the whole MongoDB is recompiled with "-sl" option (Shahriar & Haddad, 2017).

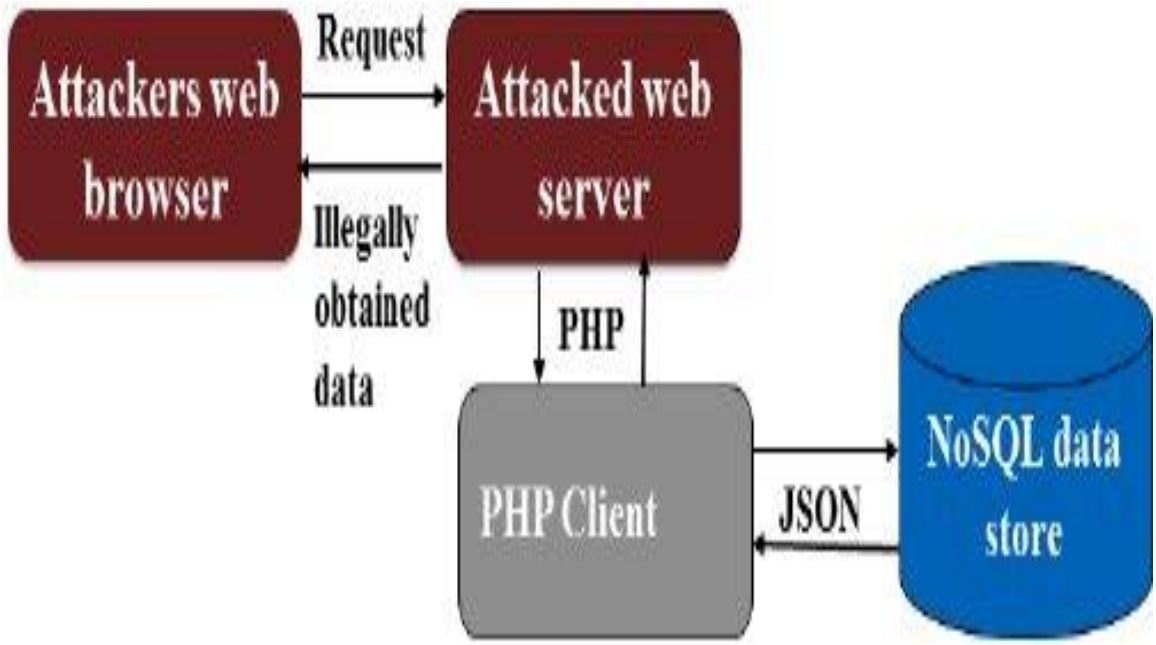
(ix) MongoDB uses a binary wire-level protocol for client interfaces on TCP port 27017, and the feature RESTful is used for managing the server on port 28017. No data encryption is performed for these ports, which means that the client-server communication is not secure (Noiumkar & Chomsiri, 2014).

(x) MongoDB version 2.4.0-2.4.4 has a vulnerability of uninitialized pointer, which allows an attacker to perform a denial of service attacks (Chahal et al., 2017).

**PHP array injection:** MongoDB databases are prone to PHP array injections because there is a built-in feature in PHP for associative arrays that allows an attacker to send malicious payloads (Aviv et al., 2015). To explain that, let us take an example of a web application that works with PHP backend that encodes the requests into JSON format. This format is then used to query the MongoDB data store.

**Figure 8**

*Architecture of a PHP web application*




*Note*: A representation of the architecture of a PHP web application (Aviv et al., 2015)

If the PHP application needs authorization through username and passwords and they are sent from an HTTP POST from the user's web browser, then the POST payload would be:

*Username=sindhu&password=Scsu*

The PHP client would then process and query the MongoDB database as:

*db- >logins -*

*>find(array("username"=>$_POST["username"],"password"=>$_POST["password"]));*

This is same as the following in Mongo query language:

*db.logins.find({ username: 'sindhu', password: 'Scsu' })*

The issue is that PHP allows sending the following malicious payload:

*username[$ne]=1&password[$ne]=1*

PHP would encode this in JSON as:

*Array("username" => array("$ne" => 1), "password" => array("$ne" => 1));*

This is encoded in mongo query as:

*db.logins.find({ username: { $ne: 1 }, password: { $ne: 1 } })*

The $ne operator in MongoDB is a "not equals" condition. This means that all the entries in the collections called logins, where usernames not equal to 1 and passwords not equal to 1, will be returned.

**JavaScript injection:** Some operations in MongoDB are vulnerable and allow an attacker to run arbitrary JavaScript expressions in place of the user input on the server. Some of these operators are $Where, Map-reduce, group, and db.eval(). When the attack string is evaluated, concatenated, or parsed into NoSQL API calls, the NoSQL injection attacks will be executed. The $Where operator is especially vulnerable because it operates as a filter in the SQL query. It can take in sophisticated JavaScript functions in order to filter the data. The attacker can pass arbitrary code into the $Where operator as a part of the query.

**Security issues or vulnerabilities in Cassandra:**

(i) Cassandra has a weak authentication. There is no authentication and authorization between the client and the Cassandra cluster by default. When a malicious user with access to the network bypasses the client authentications, then the user can extract data (Shahriar & Haddad, 2017).

(ii) The data stored in Cassandra is not encrypted in open-source version. The data is not encrypted since there is no automatic mechanism in Cassandra to encrypt the data files. So, if an attacker accesses the data, he/she can directly extract the data since the data is in plain text (Dadapeer et al., 2016).

(iii) Cassandra does not provide encryption for communications that take place between the database and its clients. If an attacker tries to monitor the network traffic, then he will be able to get all the data that is being transmitted in the network. It is also easy for the attacker to get the credentials of the users since the username and password of the user are is transmitted in the network as plain text (Dadapeer et al., 2016).

(iv) Cassandra uses Cassandra Query Language(CQL), but it is vulnerable to injection, just like SQL (Dadapeer et al., 2016).

(v) Cassandra does not have a time out a mechanism for inactive connections. Even though there are connections that are inactive, Cassandra does not close the connections for those clients. This is a vulnerability for a denial of service. An attacker will be able to make fake connection attempts, which consumes resources and makes the server unavailable for the new client connections (Noiumkar & Chomsiri, 2014).

(vi) Passwords stored in Cassandra uses MD5 hash. The MD5 hashing algorithm is a basic technique that is not cryptographically secure enough (Dadapeer et al., 2016).

(vii) Cassandra's open-source version does not support inline auditing or logging (Shahriar & Haddad, 2017).

(viii) Cassandra has an authorization mechanism called IAuthority, which comes into play when there is a read or write on each column or when a keyspace is modified. IAuthority has two implementations, which are: A pass-through and SimpleAuthority. The pass-through implementation gives full permissions to all users, and SimpleAuthority uses a flat-file that has a list of usernames and the associated permissions (Dadapeer et al., 2016). The security issues with these are:

- The authorization is implemented only on existing column families, and hence for newly added columns and column families, there is no security.
- SimpleAuthority does not reload the flat-file after every access, which means that the Cassandra process needs to be restarted before changing the effective permissions.
- The permissions that are granted to a user are based on the flat file stored on the cluster member to which the connection is established, and hence if the files for all cluster members in the cluster are not synchronized, it can be a security issue.

(ix) Cassandra has an authentication mechanism called IAuthenticate. It has two implementations, which are: default implementation and SimpleAuthenticator (Dadapeer et al., 2016). The default implementation turns off the database authentication requirement, and the SimpleAuthenticator allows you to set up a list of users and associated passwords using a flat-file. The passwords can either be stored in plain-text or in hashes using the MD5 hashing algorithm. The security issues with these are:

- Even though the passwords are stored as MD5 hashes, the communications between the database and the clients involve sending the password in plain-text. An attacker that can sniff the network can easily find out the password for authentication.
- The MD5 hashing algorithm is not considered secure anymore because of the available rainbow tables and pre-calculated lists online that can match a hash to the associated plain-text.

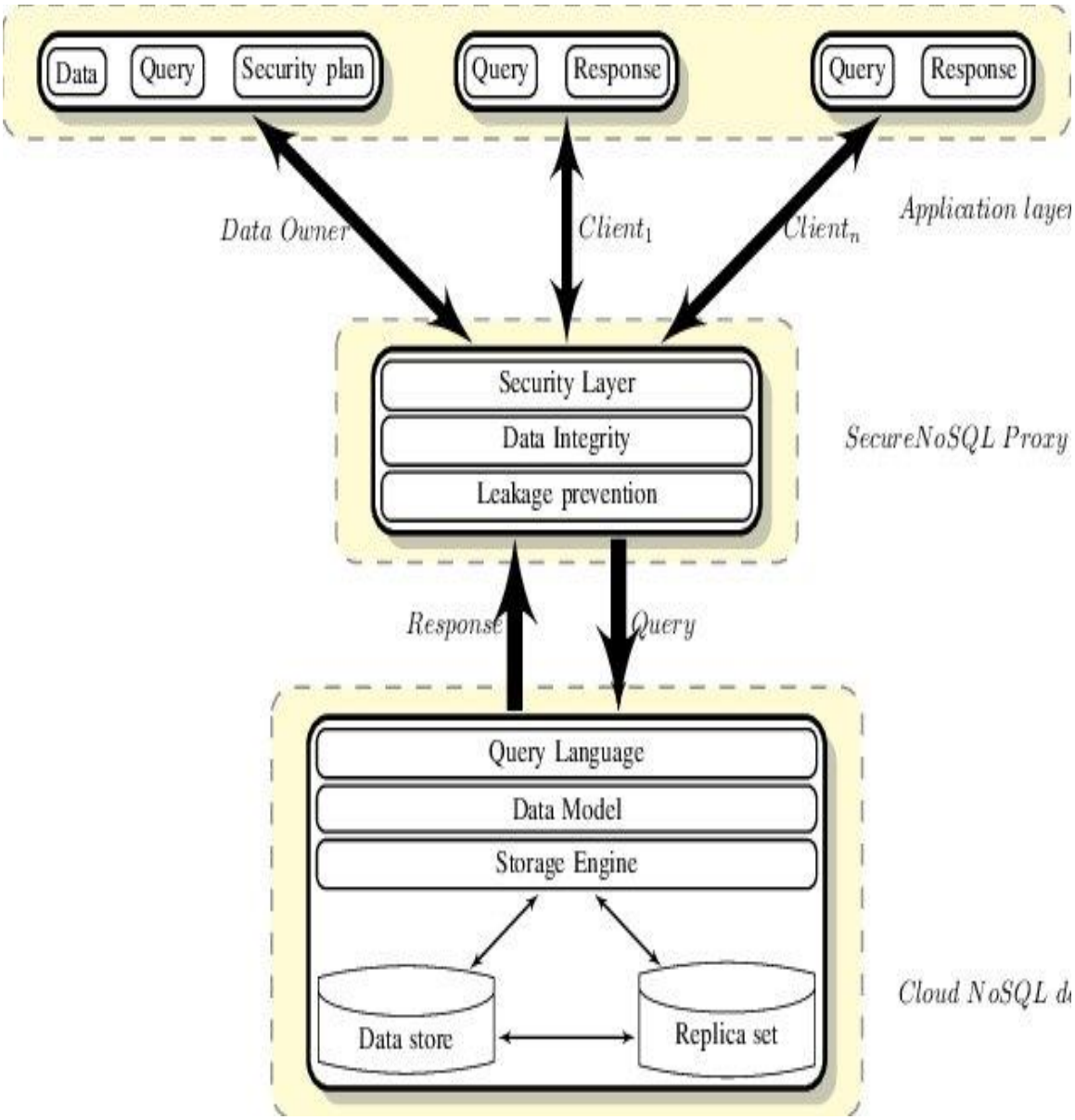
**Recommendations provided by Other Researchers:**

(Ahmadian, 2017) in "Secure query processing in cloud NoSQL" has proposed a security scheme named "SecureNoSQL", which secures querying over encrypted cloud NoSQL databases. The paper also introduces a security plan using a descriptive language based on JSON notations. The security plan describes the security parameters and maps the crypto-modules to the data elements. The architecture of the proposed SecureNoSQL is shown in the below figure.

SecureNoSQL acts as a secure proxy that allows access to the cloud server and uses cryptographic techniques for the query, response, and encryption/decryption of data. In the system, the applications on client-side issue JSON queries, the SecureNoSQL proxy encrypts and decrypts the query based on the security plan and the unmodified NoSQL DBMS processes the server-side query.

**Figure 9**

*Architecture of SecureNoSQL*

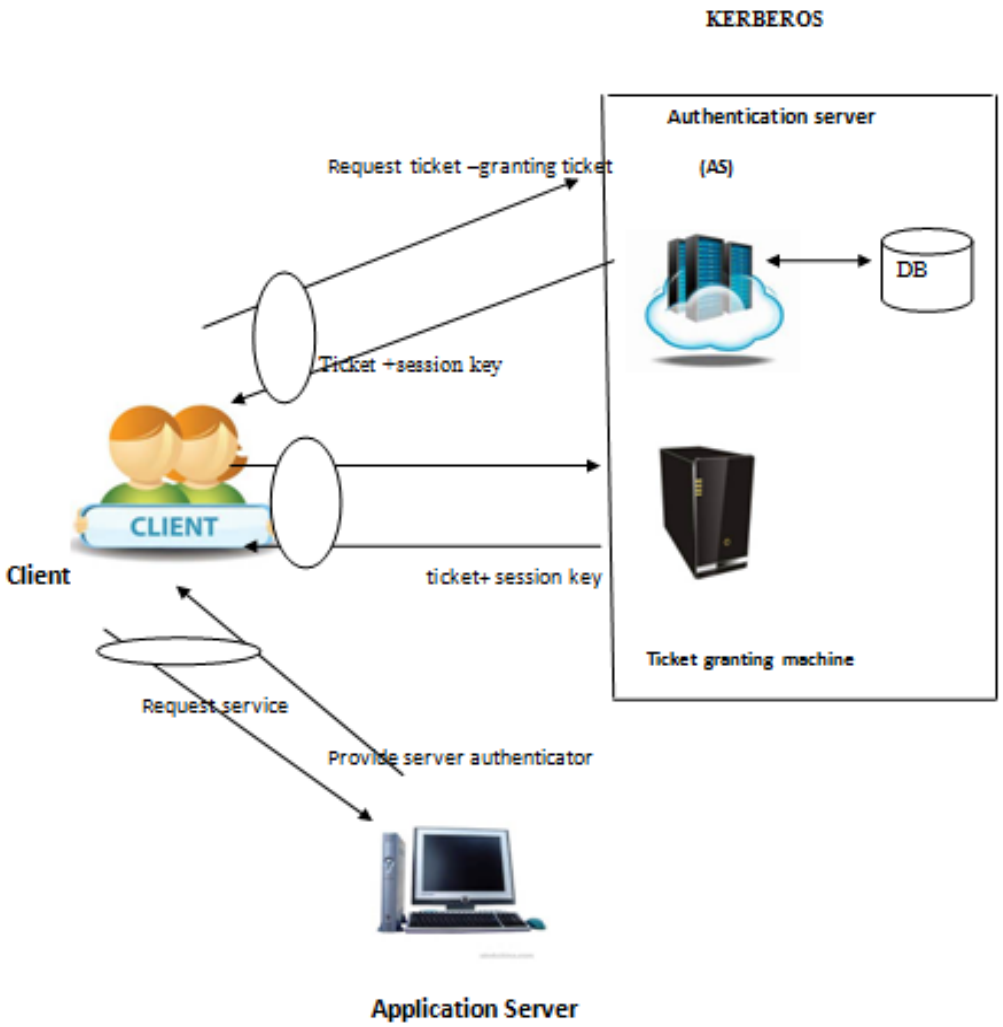



*Note:* A depiction of the architecture of SecureNoSQL (Mohammad Ahmadian, 2017)

(Priyadharshini & Rajmohan, 2017) in “Analysis on database security model against NoSQL injection” have described the model of NoSQL attack in databases, the NoSQL injection vulnerabilities in MongoDB, and Cassandra, and they have proposed an architecture to secure the NoSQL databases against NoSQL injections as shown in the below figure. In their proposed architecture, they have suggested using the Kerberos authentication protocol. They have also provided an algorithm to explain how the architecture works and have suggested extending Kerberos to provide auditing services to provide additional security.

**Figure 10**

*Proposed Architecture*




*Note:* A representation of proposed architecture proposed an architecture to secure the NoSQL databases against NoSQL injections (Priyadharshini & Rajmohan, 2017)

(Karavasilev & Somova, 2018) in “Overcoming the security issue in NoSQL databases” have discussed some of the NoSQL security issues in general such as lack of authorization features, transport encryption and client drivers, lack of built-in database encryption features, NoSQL injection and CSRF attacks, cluster desynchronization issues and virtualization leaks and disk theft risks. They have also suggested remedies to mitigate risks. They have practically analyzed the MongoDB database security and suggested enforcing authorization, performing auditing, sanitizing input data, encrypting communications, limiting network exposure, and applying data storage encryption. They have evaluated the costs in terms of performance and storage that result from implementing end to end encryption.

(Colombo & Ferrari, 2015) in “Enhancing MongoDB with fine-grained context-aware access control” have stated the drawbacks of MongoDB’s role-based access control (RBAC) model as having poor granularity level of access control and an absence of enforcement mechanisms that are context-aware. They have suggested enhancing the RBAC model along with proper support for fine-grained policies that are context-aware and also developing an enforcement monitor that is efficient. They have presented a research road map that they plan to follow in order to integrate the proposed context-aware fine-grained access control features into the MongoDB.

(Cuzzocrea & Shahriar, 2017) in “Data masking techniques for NoSQL database security: A systematic review” have given an overview of the security vulnerabilities of MongoDB and Cassandra NoSQL databases along with examples for the attacks. They have explained how useful data masking can be to protect sensitive information in the database. They have explained

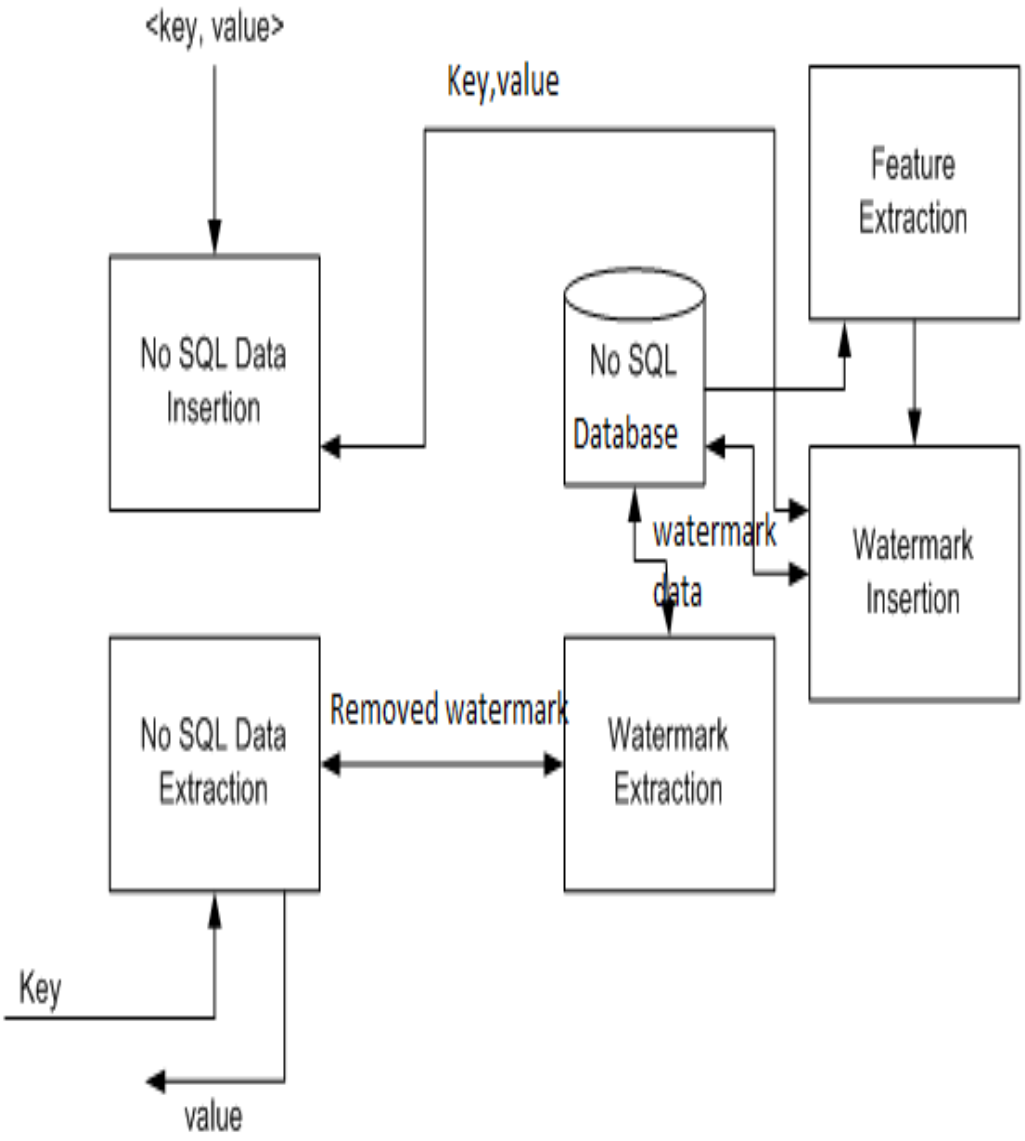
the five principles of data masking, which must be taken into account when developing a data masking technique. They have also explained two popular types of data masking architectures: In-Situ data masking architecture and on the fly server-to-server architecture. They have also discussed several different types of data masking techniques such as substitution, shuffling, number and date variance, nulling out or deletion, masking out, hashing, encrypting files and documents, and encrypting computers that may be used to secure the data. They have concluded that it is difficult to secure NoSQL databases while in operation, and hence they require additional security provided by data masking and policies in order to secure the data stored in such databases.

(Amreen & Dadapeer, 2016) in "A survey on robust security mechanism for NoSQL databases" have presented a reversible watermarking algorithm to secure NoSQL databases. The reversible watermarking algorithm has already been proposed and used for relational databases, which used histogram expansion. This technique was not robust enough to guard against heavy attacks. The authors propose an algorithm that provides appropriate watermark bandwidth that would ensure good robustness. They have provided an overview of the prediction error expansion watermarking technique and the difference expansion watermarking technique. The main purposes of the proposed algorithm are identifying theft of data, data alterations, and ensuring the right of ownership. The architecture for the proposed algorithm is shown in the below figure, where data in the form of a key-value pair is stored in a table, and the HMAC-SHA1 algorithm is used to calculate a unique watermark. The watermark is then embedded with the data, and a different table is used to store the embedded watermark and data. The same

algorithm is then used to calculate a new watermark. In order to check for any security violation, the old watermark is extracted from the data and compared with the new watermark. Four modules are used in this architecture: NoSQL data insertion, NoSQL data extraction, feature extraction, and watermark insertion.

**Figure 11**

*System Architecture*

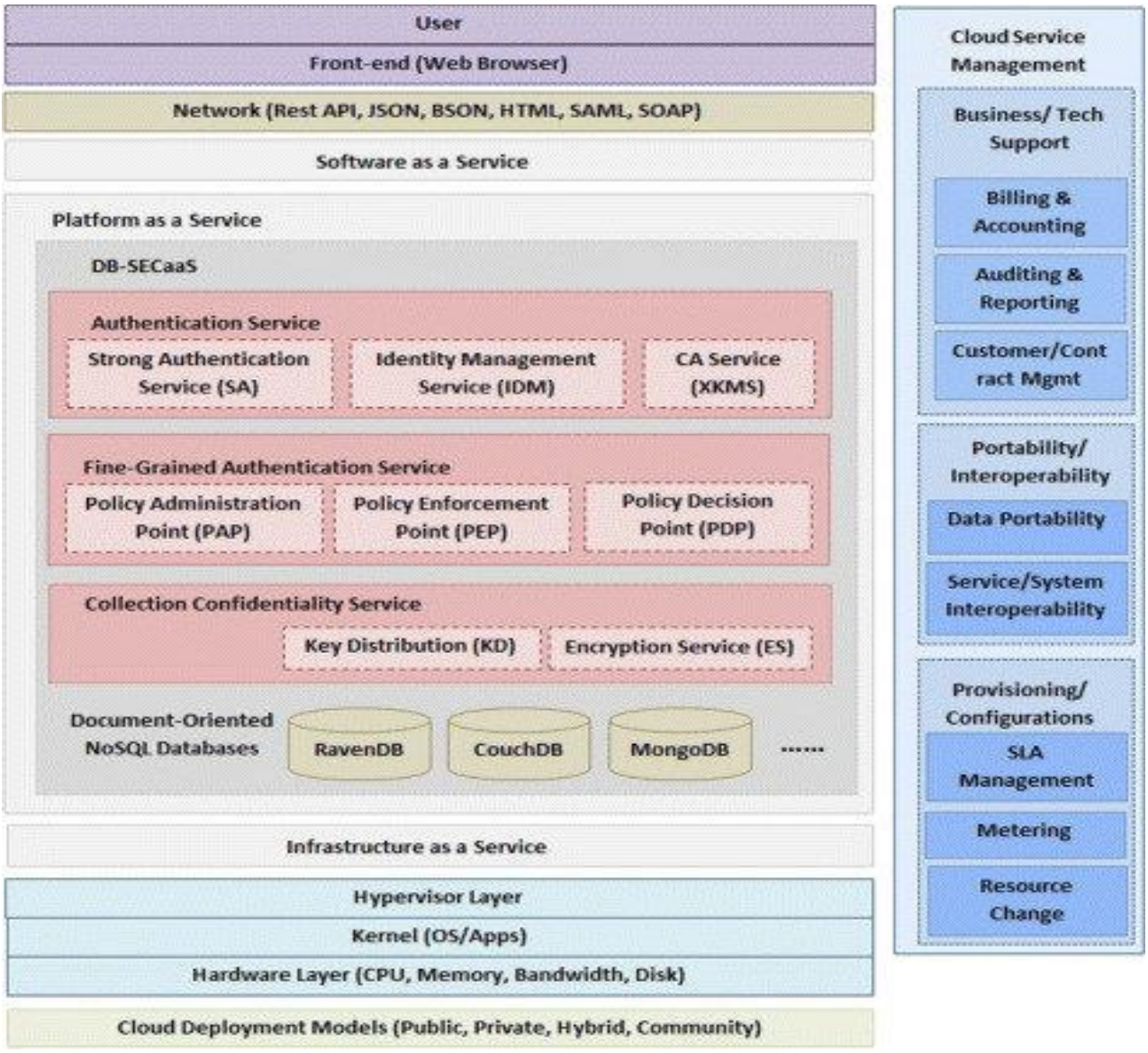



*Note:* System architecture for reversible watermarking algorithm (Amreen & Dadapeer, 2016)

(Ghazi, Masood, Rauf, Shibli, & Hassan 2016) in “DB-SECaaS: a cloud-based protection system for document-oriented NoSQL databases” have proposed a database security-as-a-service (DB-SECaaS) system for document-based databases hosted in the cloud. The architecture of the prosed system is in the below figure. The system provides authentication, fine-grained authorization, and encryption of database objects while making sure that data access is provided to authorized users in a strict need to know basis. In the system, the identities of database users and inter-system requesting parties are done using the authentication service, which includes strong authentication (SA), Identity management (IDM), and Certificate authority(CA) services. The fine-grained authorization service is used to protect data from unauthorized access. The services include policy administration point (PAP), Policy enforcement point (PEP), and Policy decision point (PDP). The data in the system is encrypted through a collection key before being stored in the collection. This is done using the collection confidentiality service. This service includes a key distribution service and encryption service. The authors have also done an evaluation of the services using NIST standards and a proper analysis of the proposed system using the Scyther model checker.

**Figure 12**

*The proposed architecture of DB-SECaaS system*



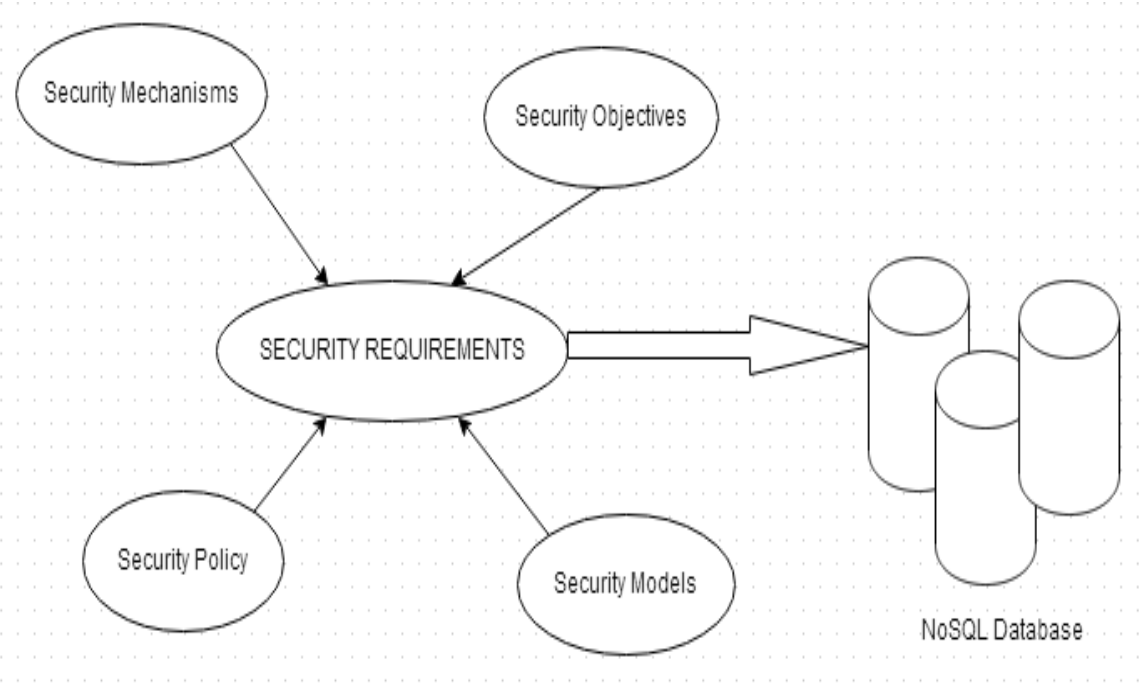

*Note.* The proposed architecture of DB-SECaaS system over a document-oriented database hosted in the cloud (Ghazi et al., 2016)

(Kadebu, Prudence, & Mapana, 2014) in "A security requirements perspective towards a secured NoSQL database environment" have discussed various security issues in general for

NoSQL databases in great detail. According to the paper, several elements need to be combined to achieve NoSQL database security. They have portrayed these elements by using a model for the NoSQL database security, as shown in the below figure. The security mechanisms that the proposed are firewalls, logging, and auditing, authentication, input validation, access control, segregation of duties, and encryption. They have described these in detail in the paper.

**Figure 13**

*Security elements for NoSQL database*




*Note*. A depiction of security elements for NoSQL database (Kadebu, Prudence & Mapana, 2014)

**Summary**

This chapter has discussed the different types of NoSQL databases, and a brief introduction of MongoDB and Cassandra databases. This chapter has discussed the literature related to the security vulnerabilities of NoSQL databases and the technical solutions proposed by other researchers. To my best knowledge, there is no research published that has a secure architecture that is specific to MongoDB and Cassandra databases. Most papers discuss the security of NoSQL databases in general, vulnerabilities in different NoSQL databases, and propose secure architectures for NoSQL databases. These papers helped me to get an overview of these topics.

# Chapter III: Methodology

## Introduction

This chapter covers the design of the study, the method for information collection, the software environment, and the methodology that will be used to solve the problem.

## Design of the Study

My plan is to use a qualitative approach to learn all the vulnerabilities and security features of MongoDB and Cassandra databases in detail and research security mechanisms that can be applied for MongoDB and Cassandra databases. This knowledge would help me come up with ideas for security recommendations that will be used to create a step by step guide for securing MongoDB and Cassandra databases.

## Information Collection

The information collected in this study is from various different research papers, conference papers, white papers, journals, books, and relevant websites. This data helped me to understand the various different aspects needed to solve the problem.

## Software Environment

To learn how MongoDB and Cassandra work in-depth and to get well acquainted with their architecture, I would need the following software:

(i) VMware Workstation (version 12)

(ii) Ubuntu (18.10) iso an image. Two virtual machines will be created using this image. One will be used to install MongoDB, and the other will be used to install Apache Cassandra.

**Methodology:**

To solve the problem, I will use the following steps:

**Step 1:** Identify and analyze all the security vulnerabilities of MongoDB and Cassandra databases.

**Step 2:** List the security considerations for each database and write the associated status for that. The main considerations would be data files, authentication, authorization, auditing, injection attacks, and client-server communication. The associated status would explain what security features are available, are not available, or are not robust enough, etc. The following is an example of what it would look like:

**Table 1**

*Considerations and statuses*

| Consideration | Status |
|---|---|
| Data files | Not encrypted |
| Client-Server communication | Not encrypted |
| Authentication | The available feature is not robust enough |
| Authorization | The available feature is not robust enough |
| Auditing | Not available |
| Injection attacks | possible |

*Note.* An example of Considerations and Statuses

**Step 3:** Based on the considerations and the statuses, I will come up with a step by step recommendations that lists the steps that developers can follow in small companies and organizations to secure the open-source versions of MongoDB and Cassandra databases.

## Summary

The information collection approach and the methodology with a three-step mechanism for the study have been identified and defined. The next chapter covers the definitions of all the considerations and statuses.

# Chapter IV: Data Presentation and Analysis

## Introduction

This chapter covers the definitions of considerations and associated statuses used in the study.

## Data Presentation

The considerations and the associated statuses that were identified to get the overall breadth of security features in MongoDB and Cassandra databases are explained below. These considerations and statuses are later used in the study to evaluate the security of these databases.

1. **Data Files:** Data files are operating system files that are used to store data within a database or a computer system.

   Data files need to be properly secured with encryption to prevent information theft and intentional corruption by an attacker.

   **Statuses:**

   - Encrypted: The data in the data files is encoded by converting plain text to ciphertext.

- Not encrypted: The data in the data files is not encoded, which means it is stored in plain text. Hence if an attacker gets his/her hands on the data, they can make malicious use of the information.

2. **Client-Server Communication:** Client-Server communication is a process where clients (a program) send requests for services or resources to the server (another program), and the server responds back to the client requests (Sullivan, 2019). In most cases, there are multiple clients and a single server.

   Securing the communications between clients and servers is very important because client-server communications also involve the exchange of credentials when authentication is taking place. If the communications are not encrypted, an attacker monitoring the network traffic can get a hold of the information and use it for malicious purposes. Also, the clients and servers need to be authenticated using a protocol such as TLS (Transport layer security) to guarantee the integrity and confidentiality of the information that is exchanged.

   **Statuses:**

   - Encrypted: The data that is exchanged between the Clients and the Server is encrypted.
   - Not Encrypted: The data that is exchanged between the Clients and the Server is not encrypted.

3. **Authentication:** Authentication is the process of validating that only an authorized person is given access to the database (Chahal et al., 2017).

   A weak authentication mechanism can expose the database to replay attacks or man-in-the-middle attacks

   **Statuses:**

   - The available feature is robust: The available authentication feature is strong and is very hard to bypass
   - The available feature is not robust enough: The available authentication feature is weak and is easy to bypass and is vulnerable to attacks.

4. **Authorization:** Authorization is a process of giving permission to users to access the data depending on their role (Chahal et al., 2017).

   A lack of authorization features compromises the overall application security and is a loophole for hostile access from an attacker.

   **Statuses:**

   - The available feature is robust: The available authorization feature is strong and is very hard to bypass
   - The available feature is not robust enough: The available authorization feature is not strong enough.

5. **Auditing:** Data auditing is a process designed to let an administrator understand who looked at what and when who had changed what and when (Yehuda, 2018). It provides a way to log user activity occurring on a database.

   Many companies and organizations have internal security policies and external mandates that require auditing. Hence, auditing is a very important tool that can be used to investigate what happened if an attack were to happen.

   **Statuses:**

   - Available: Auditing features are available in the open-source version.
   - Not available: Auditing features are not available in the open-source version.

6. **Injection attacks:** NoSQL injection is a security vulnerability where an attacker makes malicious use of user input to take control of the database queries, which in turn compromises the databases. Using this technique, an attacker can expose the unauthorized information, make changes to the data, escalate the privileges, or take down the whole application.

   **Statuses:**

   - Possible: The database is not very secure against injection attacks.
   - Very difficult: The database is well secured against injection attacks.

## Data Analysis

A qualitative method was used to analyze all the vulnerabilities and come up with a set of considerations and statuses that best describe the security issues with open source versions of MongoDB and Cassandra databases. These considerations and statuses were further analyzed in a qualitative manner to come up with a step by step recommendations that can be used to secure the open-source versions of these databases.

## Summary

This chapter covered a detailed description of the considerations and the associated statuses used in the study. The results of the study are explained in the next chapter.

# Chapter V: Results, Conclusion and Recommendations

## Introduction

This chapter clearly identifies and analyzes the security vulnerabilities of MongoDB and Cassandra databases and gives step by step recommendations to secure the open-source versions of these databases.

## Results

After following the methodology discussed in the previous chapter, I came up with the following considerations and statuses for Cassandra and MongoDB databases:

## Considerations and Statuses

### *Cassandra*

**Table 2**

*Cassandra's considerations and statuses*

| Consideration | Status |
|---|---|
| Data files | Data in storage is not automatically encrypted. It is stored in plain text by default. |

**Table 2 Continued**

| | |
|---|---|
| Client-Server communication | Not encrypted. An attacker can monitor the database traffic to see all communication. |
| Authentication | The available feature is not robust enough. The authentication is turned off by default. Using SimpleAuthenticator, users and passwords can be set with a flat-file with a password in MD5 hash, but the password is still transmitted in plain text by the client interface. |
| authorization | The available feature is not robust enough. The IAuthority interface allows full permissions to all users and the SimpleAuthority uses a flat-file and not a maintained file across the cluster. |

**Table 2 Continued**

| | |
|---|---|
| auditing | Not available. Inline auditing is not supported. |
| Injection attacks | Possible. Cassandra query language is a parsed language vulnerable to injection attacks. |

*Note.* A list of Cassandra's Considerations and Statuses

**MongoDB**

**Table 3**

*MongoDB's considerations and statuses*

| Consideration | Status |
|---|---|
| Data files | Data in storage is not automatically encrypted. It is stored in plain text by default. |
| Client-Server communication | Not encrypted. An attacker can monitor the database traffic to see all communication. |
| authentication | The available feature is not robust enough in standalone mode. In the Sharded mode, authentication is not supported. |

**Table 3 Continued**

| | |
|---|---|
| authorization | The available feature is not robust enough. A basic role-based access control model is supported, but the access control is enforced at an inappropriate granularity level. |
| auditing | Not available in open-source version |
| Injection attacks | Possible. The internal scripting language is JavaScript, which is an interpreted language with a potential for injection attacks. |

*Note.* A list of MongoDB's considerations and statuses

Based on the above security considerations and statuses for each database, I came up with the below step by step recommendations for securing the open-source versions of MongoDB and Cassandra databases:

## Step by Step Recommendations

### MongoDB

**Step 1:** The access control is not enabled by default in MongoDB. When initializing the MongoDB shell, the –auth keyword can be used to enable authorization. Setting up authorization reduces the risk of account breaches. To create users with roles, the following command can be used:

```
Use admin
Db. createUser(
{
User: “sindhu”,
Pwd: “securepassword”,
roles: [{role:”useAdminAnyDatabase”,db:”admin”}]
}
)
```

**Step 2:** In most cases, hackers first scan the default port numbers before they attack (Paramathmuni, 2018). Hence, change the default port numbers in the MongoDB configuration file: mongo.config

**Step 3:** Authentication can be enabled by navigating to the #security section in the MongoDB configuration file “Mongod.conf”. Remove the “#” in front of security to enable it.

Security:

Authorization: “enabled”

Restart MongoDB now

To test the authentication, the show dbs command can be used. If the authentication worked, an error should show up like below:

**Figure 14**

*Error that indicates that the authentication worked*

```
Output
2017-02-21T19:20:42.919+0000 E QUERY    [thread1] Error: listDatabases failed:{
        "ok" : 0,
        "errmsg" : "not authorized on admin to execute command { listDatabases: 1.0 }",
        "code" : 13,
        "codeName" : "Unauthorized"
 . . .
```

*Note.* An example of an error that indicates that the authentication worked

**Step 4:** Automated scripts can detect MongoDB instances that are not protected by a firewall. Verify the status of the firewall using the command:

sudo ufw status

If the status says inactive, activate it using the following command:

host$ sudo ufw enable

Also, make allow SSH using the command:

host$ sudo ufw allow OpenSSH

The output should indicate that only OpenSSH is allowed:

**Figure 15**

*Output when only OpenSSH is allowed*

```
Output
Status: active

To                         Action      From
--                         ------      ----
OpenSSH                    ALLOW       Anywhere
OpenSSH (v6)               ALLOW       Anywhere (v6)
```

*Note.* An example of output when only OpenSSH is allowed

**Step 5:** If remote access needs to be allowed, we can restrict that access to a specific host for the default port 27107 using the following command:

host$ sudo ufw allow from client_ip_address to any port 27107

For each additional client who needs access, re-runs this command using the IP address.

**Step 6:** A replication keyfile can be enabled to automatically enable authentication and ensure data encryption. Using this method, only hosts that have this file installed

will be able to join the replica set. A keyfile can be generated using any preferred method. Once it is generated, copy the keyfile to the replica set members, enable the access control and start the replica set

In order to enable the replication keyfile, add the following to the MongoDB configuration file (mongo.conf):

Security:

Keyfile: <path to keyfile>

**Step 7:** Although auditing features are available in some versions of MongoDB, there are none available for the open-source version. Also, there is no third-party tool that can be installed in the MongoDB open-source version to generate audit logs. This can be an improvement in the future where MongoDB releases auditing features for the open-source version, or a third-party tool is developed that can generate audit logs for the open-source version of MongoDB.

**Step 8:** In order to prevent any injection attacks, a RESTful API can be developed that connects to the database with a limited account only. In addition, the data input can be sanitized, and strong authentication can be used. Also, allow only direct connections from the API and network or system firewalls that can be used to block all the native client communications.

**Cassandra**

**Step 1:** Cassandra does not automatically encrypt the data.

We can enable inter-node encryption by navigating to the server_encryption_option section in Cassandra.yaml. The default for internode_encryption is set as none. Change this to either rack, dc, or all.

**Step 2:** We can also enable Client to Node Encryption by navigating to the client_encryption_options section in Cassandra.yaml. The two primary options for enabling encryption here are “enabled” and “optional”.
If both are set as false, the client connections are unencrypted. ‘
If both are set as true, the same port supports both encrypted and unencrypted connections.
If enabled is set as true and optional is set to false, all the client connections are then secured. Choose this option for better security.

**Step 3:** In Cassandra.yaml file, turn the authentication option from AllowAllAuthenticator (default authentication which does not perform any authentication checks and requires no credentials) to PasswordAuthenticator, that can be used to enable username and password authentication.

Authenticator: PasswordAuthenticator

Restart the node after this.

**Step 4:** In order to prevent any security breaches, change the default superuser, which is ‘Cassandra’ to another superuser:

CREATE ROLE <new_super_user> WITH PASSWORD = '<provide a strong password here>'
AND SUPERUSER = true
AND LOGIN = true;

**Step 5:** Authorization can be enabled by changing the authorizer setting in the Cassandra.yaml file. By default, it is set as AllowAuthorizer. This setting grants all permissions to all the roles. Change this to CassandraAuthorizer, which allows for full permissions management.

Authorizer: CassandraAuthorizer

Once authorization is turned on, statements such as GRANT PERMISSION, REVOKE PERMISSION, etc. can be used to set the access privileges for the clients. Restart the node after this.

**Step 6:** As auditing features are only available for the enterprise versions, a third-party tool such as ecAudit can be installed to get the auditing functionality for the open-source version of Apache Cassandra.

Depending on the version of Cassandra, a compatible ecAudit version can be installed. ecAudit requires a JVM that is Java 8 compatible.

To begin the setup, put the ecaudit jar file in the directory: $CASSANDRA_Home/lib/ directory

In order to enable the plug-in, the following settings need to be changed in Cassandra.yaml file:

Authenticator: com.ericson.bss.cassandra.ecaudit.auth.AuditAuthenticator

Authorizer: com.ericson.bss.cassandra.ecaudit.auth.AuditAuthorizer

Role manager: com.ericson.bss.cassandra.ecaudit.auth.AudiRoleManager

All the audit logs are stored in the audit.yaml file in Cassandra's configuration directory.

## Conclusion

This study provides and discusses a comprehensive list of vulnerabilities of NoSQL databases in general and vulnerabilities that are specific to MongoDB and Cassandra databases. The study also identifies and describes a set of security considerations for each database and provides recommendations that can be used to secure the MongoDB and Cassandra databases.

## Future Work

The research work done for this study can be expanded further to help make the open-source versions of NoSQL databases such as MongoDB and Cassandra more robust in security like their RDBMS counterparts.

## References

Ahmadian, M. (2017). Secure query processing in cloud NoSQL. 2017 IEEE International Conference on Consumer Electronics (ICCE). doi:10.1109/icce.2017.7889242

Amreen, & Dadapeer. (2016). A survey on robust security mechanism for NoSQL databases. International Journal of Innovative Research in Computer and Communication Engineering,4(4), 7662-7666. doi:10.15680/IJIRCCE.2016.0404265

Aviv, R., Shulman-Peleg, A., & Bronshtein, E. (2015). NO SQL, No Injection? Examining NoSQL Security.

Chahal, D., Kharb, L., & Gupta, M. (2017). Challenges and Security issues in NoSQL databases. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*,*2*(5).

Cimpanu, C. (2018, August 07). California Voter Database Compromised in MongoDB Incident. Retrieved from https://www.bleepingcomputer.com/news/security/california-voter-database-compromised-in-mongodb-incident/

Colombo, P., & Ferrari, E. (2015). Enhancing NoSQL datastores with fine-grained context-aware access control: A preliminary study on MongoDB. International Journal of Cloud Computing,6(4), 292. doi:10.1504/ijcc.2017.090197

Cooke, A. (2018, May 08). Top Rated NoSQL Databases for 2018 I TrustRadius. Retrieved from https://www.trustradius.com/buyer-blog/top-rated-nosql-databases-2018/

Cuzzocrea, A., & Shahriar, H. (2017). Data masking techniques for NoSQL database security: A systematic review. 2017 IEEE International Conference on Big Data (Big Data). doi:10.1109/bigdata.2017.8258486

Dadapeer.,Indravasan, M., & G, Adarsh. (2016). A Survey on Security of NoSQL Databases. International Journal of Innovative Research in Computer and Communication Engineering,4(4). doi:10.15680/IJIRCCE.2016. 0404194

Dayley, B. (2014, September 18). Informit. Retrieved from http://www.informit.com/articles/article.aspx?p=2247310

Dindoliwala, V. J., & Morena, R. D. (2017). Survey on Security Mechanisms In NoSQL Databases. International Journal of Advanced Research in Computer Science,8(5).

Fedak, V. (2018, March 02). Top 5 reasons to use the Apache Cassandra Database – Towards Data Science. Retrieved from https://towardsdatascience.com/top-5-reasons-to-use-the-apache-cassandra-database-d541c6448557

Ghazi, Y., Masood, R., Rauf, A., Shibli, M. A., & Hassan, O. (2016). DB-SECaaS: A cloud-based protection system for document-oriented NoSQL databases. EURASIP Journal on Information Security,2016(1). doi:10.1186/s13635-016-0040-5

Hou, B., Qian, K., Li, L., Shi, Y., Tao, L., & Liu, J. (2016). MongoDB NoSQL Injection Analysis and Detection. 2016 IEEE 3rd International Conference on Cyber Security and Cloud Computing (CSCloud). doi:10.1109/cscloud.2016.57

Kadebu, Prudence., & Mapana, I. (2014). A security requirements perspective towards a

secured NoSQL database environment. 2014 International Conference of Advance Research and Innovation (ICARI)

Karavasilev, T., & Somova, E. (2018). Overcoming the security issue in NoSQL databases. 2018 TechSys conference

McCallion, J. (2017). 26,000 unsecured MongoDB servers hit by ransomware. (2017, September 06). Retrieved from https://www.itpro.co.uk/security/27885/26000-unsecured-mongodb-servers-hit-by-ransomware

Noiumkar, P., & Chomsiri, T. (2014). A comparison the level of security on Top 5 open source NoSQL databases. 2014 The 9th International Conference on Information Technology and Applications

Okman, L., Gal-Oz, N., Gonen, Y., Gudes, E., & Abramov, J. (2011). Security Issues in NoSQL Databases. 2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications. doi:10.1109/trustcom.2011.70

Paramathmuni, P. (2018, October 31). MongoDB security tips. (2018, October 31). Retrieved from developer.rackspace.com website: https://developer.rackspace.com/blog/MongoDB-Security-Tips/

R, Rajmohan., & S, P Priyadharshini. (2017). Analysis on Database Security Model Against NOSQL Injection. l Journal of Scientific Research in Computer Science, 2(2). Retrieved from http://ijsrcseit.com/paper/CSEIT172229.pdf

Sadalage, P. (2014, October 3). NoSQL Databases: An Overview. Retrieved from https://www.thoughtworks.com/insights/blog/nosql-databases-overview

Saran, R., M, Sai Baba., S, Jayanthi., & E, Soundararanjan. (2015). Storing of Unstructured data into MongoDB using Consistent Hashing Algorithm. International Journal of Emerging Technologies in Engineering Research,3(3).

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

Sullivan, J. (2019, April). client-server model (client-server architecture). Retrieved from searchnetworking.techtarget.com website: Retrieved from searchnetworking.techtarget.com website: https://searchnetworking.techtarget.com/definition/client-server

Vishwakarma, R. (2017, May 16). The Different Types of NoSQL Databases. Retrieved from https://opensourceforu.com/2017/05/different-types-nosql-databases/

Yehuda, yaniv. (2018, March 21). Database Audits: Why You Need Them and What Tools to Use. Retrieved from www3.dbmaestro.com website: https://www3.dbmaestro.com/blog/database-audits-why-you-need-them-what-tools-to-use

Zahid, A., Masood, R., & Shibli, M. A. (2014). Security of sharded NoSQL databases: A comparative analysis. 2014 Conference on Information Assurance and Cyber Security (CIACS). doi:10.1109/ciacs.2014.6861323