Security Audit of NoSQL DBMS

ISSM-581: Research Methods III
Spring 2021

Supraja Kairoju (skairoju@student.concordia.ab.ca)
Rouheen Sultana (rsultan1@student.concordia.ab.ca)
Priyanka Danidharia (pdanidha@student.concordia.ab.ca)

Research Project

Submitted to the Faculty of Graduate Studies
Concordia University of Edmonton

In Partial Fulfilment of the
Requirements of ISSM-581 course

Concordia University of Edmonton
FACULTY OF GRADUTE STUDIES
Edmonton, Alberta

Advisor: Dr Sergey Butakov (sergey.butakov@concordia.ab.ca)
Department of Information Systems Security and Assurance Management
Concordia University of Edmonton,
Edmonton T5B 4E4, Alberta, Canada

Security Audit of NoSQL DBMS

Supraja Kairoju

Rouheen Sultana

Priyanka Danidharia

Approved:

Sergey Butakov [Original Approval on File]

Sergey Butakov
Primary Supervisor

Date: June 23, 2021

Patrick Kamau [Original Approval on File]

Patrick Kamau, PhD, MCIC, PChem.
Dean, Faculty of Graduate Studies

Date: June 23, 2021

Table of Contents

List of Tables

List of Figures

Security Audit of NoSQL DBMS

Supraja Kairoju (skairoju@student.concordia.ab.ca)
Rouheen Sultana (rsultan1@student.concordia.ab.ca)
Priyanka Danidharia (pdanidha@student.concordia.ab.ca)

***Abstract* – The necessity to store extensive volumes of data as the information produced increases exponentially. An increasing number of businesses have embraced different types of non-relational databases in recent years, generally referred to as NoSQL databases. Security auditing procedures for these new technologies ought to be in place for minimizing the potential negative impacts from the risks associated with them. The purpose of this study is to undertake security audit on such NoSQL databases while considering the specific risks associated with them. Major findings regarding the identified security controls in NoSQL databases namely MongoDB, Redis and CouchDB are discussed, saving considerable amount of time for the database users. This enables the database developers to focus on strengthening their weaknesses when one of the NoSQL flavors is chosen. All in all, this research provides a security guideline for the organizations implementing NoSQL and aids the developer of these databases to identify relevant security risks.**

Keywords—Databases, NoSQL DBMS, Auditing, Security Audit, Security Assessment

I. INTRODUCTION

An extensible and fully featured data storage solutions are one of the major needs for virtually any business. With the evolving nature of technology, databases migrated from local to cloud implementations with the latter providing essential benefits of availability, server management, scalability, agility, collaboration, and accessibility [1]. Speed of the development and pressure to push products to the market also be one of the many reasons for which bugs are introduced [2], which implies the protection of the application with appropriate security controls and compliance to already established security frameworks, becomes paramount. This research, therefore, focuses on the implementing appropriate controls of the database security paradigm, specifically NoSQL.

Auditing is required to evaluate the effectiveness of certain prevention controls for protection of the underlying database as it allows for tracking and understanding of how the documents are used, as well as insight into any threats of exploitation or breaches. If the threats have been discovered, appropriate proactive solutions or reactive solutions or both, must be enforced to ensure that the database is safe from attack. Three NoSQL database management systems (DBMS), namely, MongoDB, Redis and CouchDB have been used in this research project to execute checklist-based audit to find associated security risks, inclusively providing and implementing remediate solutions to ensure the database flavor's security to provide reasonable assurance. This checklist-based framework can then be utilized as a base for regular audit purposes and for future research.

With the steadily increasing generation of data, the need for storage and analysis of these massive data has increased exponentially. New requirements in storage and analysis capacity facilitate shifting from traditional SQL to novel NoSQL to databases. Data industry trying to keep up with the demands and sometimes data security, which is vital to any organization, is being ignored. To ensure reasonable security of a database, compliance measures must be implemented to achieve assurance and to reduce future costs. The aim of this research is to discuss about several security shortcomings in NoSQL databases, their solutions and lastly, provide a checklist on how to ensure proper compliance with audit frameworks such as, analyze the levels of security these databases provide to ensure reasonable assurance, and develop a checklist-based framework to comply with some management systems, analyze subject discussion revolves around the levels of security these databases provide to ensure reasonable assurance, and develop a checklist-based framework to comply with some well-known audit frameworks, which can then be utilized for regular audit purposes and establishing assurance.

The need for storing as well as securing data has been exponentially increasing over the last two decades. Regular audits have become the steppingstone for maintaining database quality, compliance with privacy regulations and protection against data breaches [3]. Ensuring proper accountability, investigation of internal suspicious activity and gathering relevant information are some other use cases where an audit is a must [4]. Recent alarming breaches due to various reasons, for example 3 out of 7 in [5] were data breaches, or misconfigurations in [6] [7] - have intensified the need to establish essential audit controls and procedures in place, as well as other defences, to protect organizations from upcoming

breaches. However, according to the authors' knowledge, there is no generic security audit or overall security audit methodologies published for NoSQL Databases. Hence, this research paper shall provide some detailed insights on MongoDB, Redis and CouchDB vulnerabilities, and their security audits to achieve assurance goals related to security, availability, and compliance in these databases.

II. Literature Review

A. NoSQL

NoSQL databases were developed by global leading IT companies, with sole purpose of satisfying the need of databases coping up with increasing data processing requirements and Big Data technologies. For Big data flexibility is one of the most essential characteristics for a database, for which dynamic schema of NoSQL is perfect. Some of the major benefits of using NoSQL databases for large amount of data is 1) they offer predictive analysis. An example of this is data from various social media sites such as Facebook, Twitter, Instagram, etc. 2) NoSQL databases can perform well and give better, larger, and powerful results if needed as they are horizontally scalable [8] [9]. These advantages of working with NoSQL gives them extra edge in comparison to RDBMS and makes them more preferable choice for especially big data applications. Following are some features of NoSQL which proves to be extra beneficiary for security audit [10] [11]:

- Management of large data at high speed is easy and more efficient with the scale-out of the architecture of NoSQL.
- NoSQL works well and supports stored unstructured data, semi-structured data, and structured data equally.
- It's less complex to enable easy updates on schemas and their fields.
- NoSQL uses the cloud to its extent and delivers zero downtime.
- Due to the advanced features of NoSQL, it requires very little extract, transform, and load.
- Ability to handle change and adapt to it.
- No legacy code means support for old hardware platforms is not required.
- No demand to separate the data warehouse system and shipping a large amount of data over the network for locally combined analysis.
- User is free to select vendor as per requirement without any rules binding them. Some of the vendors available are IBM, Microsoft, and Oracle.

However, NoSQL is an emerging technology, that does not have enough material to understand its issues and increase awareness. Using these databases sometimes can cause a lot of problems related to security management, Data Consistency, Scalability (not completely scalable in all situations) etc. This research provides a list of few such problems and some recommendations to mitigate them. Also, a security audit is made to identify, assess, and implement key security controls in NoSQL Databases, that would help the new or existing NoSQL Database users to predict the risk against respective databases and to majorly focus on the protection of controls which are at higher risk.

B. Risks Associated with NoSQL

The biggest risk in the use of NoSQL is regarding security. Authentication and encryption both are very weak within its vanilla implementation. Other shortcomings include weak password storage, insufficiency of encryption support for the data files, fragility to injection and DOS attacks. Communication between server and client is, by default, in plain text format without any encryption or security layers. Furthermore, data at rest is also unencrypted making NoSQL unsuitable for use of external encryption tools like LDAP, Kerberos etc. [12] [13] [14] [15].

In addition to not always well-developed security features to external threats, NoSQL DBMSs also have internal shortcomings which result in data management risks such as the technology lacks a stable user base in comparison to relational databases which implies less overall stability than RDBMS. NoSQL also lacks deep analytic support, compatibility with hybrid clouds and support to transactional functions for the use in financial systems. Moreover, NoSQL databases support consistency and scalability but they do not support strict conformance to the Atomicity, Consistency, Isolation, and Durability (ACID) properties of a database. All these risks and shortcomings sometime makes companies reluctant to adopt NoSQL DBMS in major projects. The risks associated with NoSQL are categorized according to three states of data: i) Data in use ii) Data in motion and iii) Data at rest [16] [13] [14] [15].

1) Risks associated with Data at Rest

i) Weak Authentication and Encryption in NoSQL DBMS

Most certified NoSQL databases only have a simple access control feature, while others do not have one at all. Without any credentials or restrictions in place, this lack of protection leaves an open vulnerability to hostile access. Using default authorization credentials is a common mistake made by big data users. Therefore, vulnerabilities like an absence of proper authorization functionalities, access controls, segregation of user roles, are created. By design, search engines and significant in-memory key-value solutions have this issue [17] [18] [19] [20].

To overcome this, several defense-in-depth strategies must be implemented. For example, the user must build a RESTful API around the database solution and applying access tokens as user credentials for the Web API clients [19] [20] [17] [18]. Native authentication features can also be leveraged for solving these issues. Modern NoSQL solutions also lack built in encryption storage engines, data integrity functionality, encryption aggregate functions, and store texts plainly which imposes too many security risks. Solution to mitigate this is to

develop a transparent encryption application layer [18] [19] [20].

CouchDB has an admin user (e.g., root user or super user or administrator) that can do anything and everything while installing CouchDB. For successful starting up CouchDB it is must to create an admin user by default. To quickly authenticate CouchDB simple methods like basic authentication (method for a http user agent) is preferred. The biggest disadvantage of using basic authentication is it requires sending user credentials for each request which can easily damage operation performance (as for every request CouchDB computes the password hash) and this makes CouchDB vulnerable to strong credential abuse attacks. Setting up SSL for improvement of all types of authentication methods' security is recommended to resolve this issue [21].

ii) Weak Password Storage

MongoDB's password storage system is insecure. The password storage form is an MD5 hash of the string "username: mongo: password," which is effortlessly obtainable from the admin data-files, meaning that if an attacker obtains the MD5 hash, he or she will have access to the database's actual password. The encryption key does not shift and remains constant for all users. As a result, the encryption becomes less secure. Redis also has a password storage system that isn't very safe. Since Redis' password is set in plain text and the number of password attempts is not rate restricted or limited in any sort, brute force attacks are a major threat. The admin user or authentication is disabled by default in Cassandra, and password storage is similarly insecure. Solution for MongoDB's weak password storage is to store hashed password using Bcrypt or Node.js or Mongoose (software for hashed password generation) for security [12] [22] [23].

2) Risks Associated with Data in Motion

i) Denial of Service Attacks

In their network code, some NoSQL database servers use a Thread-Per-Client model. The NoSQL project suggests using various types of connection pooling because every link requires the NoSQL server to begin with a new thread (The network's TCP overhead is also considered in). An attacker can prevent a NoSQL server from incorporating new client connection requests by forcing it to offer all its resources to forge link attempts. An invader's only requirement is details such as the IP addresses of cluster members. The sniffers are used by the attacker to get information from the network. Since the current approach does not allow for indirect connections, any link that is opened without transmitting data requires a thread and a file-descriptor that cannot be launched [17].

ii) Quick access to ports that are open

Many of the NoSQL servers, according to a report, are exposed to the internet and can be easily used by the hackers. Someone once looked for open servers on the Internet/24 and discovered 48, which is a huge security risk. For instance, Cassandra uses port 8888 as its default port. If Cassandra server's port is known by hackers and a sufficient link is established, the Cassandra server can be stuck and shut down. However, Since Cassandra uses the Thrift Protocol, it creates one thread for each client, and since it lacks a time-out function for passive clients, the connection pool of Cassandra can quickly become occupied, causing it to crash [12].

iii) Lack of IP Binding

NoSQL databases are set to be publicly available, such as All data in MongoDB is stored on a public IP address and is available to the public. By default, Redis is open to the public, and whoever looks for its open port number: 6379 will be able to view the Redis server info. Both databases include the use of firewalls to protect the IP binding. This will restrict which entities can connect to the server and how the database can be used. It is recommended that application servers have access to the database. For Instance, Applications hosted on a cloud platform such as Azure or AWS, may use security groups to bind links to public IP addresses and ports. When other service providers are opted, 'ip_tables' should be used to secure it. Only IP addresses that can access the servers and retrieve data from the database will be stored in these tables [12].

3) Risks associated with Data in Use

i) Injection Attacks

MongoDB is the most common NoSQL database that is exposed to injection attacks. MongoDB makes heavy use of JavaScript on the server side to improve performance, and as a result, it is often vulnerable to injection attacks. The internal operator of MongoDB "$where," which was intended to be implemented as a search query like SQL's "where" clause, can also filter data using sophisticated JavaScript functions. The CQL works under Thrift and at the RPC level, so it is not vulnerable to SQL injection. While Aniello et al. indicated Cassandra has little weakness, those were not the CQL's findings. Since the Redis protocol does not support string escape, it is not at risk to injection attacks when used with a standard client library [12].

ii) Lack of proper authorization

An important issue attracting auditor focus is - data integrity, which is a set of significant evaluations of completeness, authorization, consistency of data, accuracy of data, and accurate authorization - is a liability for NoSQL. To exemplify, in pre-existing settings, authorization check is disabled in MongoDB. However, MongoDB uses a role-based approach which allows different types of solutions as users can be authorized on pre-database level. On the other hand, authorization in Cassandra is pre-approved for all the clients without any regards of credentials. Unlike traditional RDBMS, Cassandra is a schema-less database, which explains the improper data models. According to Apache official manual of Cassandra, if Cassandra Authorizer is determined, then the administrator can be allowed to have advanced privileges

including AUTHORIZE, ALTER, SELECT, DROP, CREATE, MODIFY, on any resources (TABLE, KEYSPACE, ALL KEYSPACES) to a determined user, implementing CQL (Cassandra Query Language) statements. As for Redis, the password storage method lacks proper security and protection. Since there are no restrictions or rate limit on the number of password trials and password is saved in clear text, Redis faces major threats from brute force attacks [17] [20] [24].

iii) Risks and technical challenges associated with migrating from SQL to NoSQL technology

While achieving high performance and high scalability NoSQL sacrifices one or more of the following properties: atomicity, consistency, isolation, and durability. ACID properties can guarantee correctness of data based on the features such as triggers, constraints setup, keys, and triggers which is a lost cause in NoSQL. Majority of NoSQL solutions (except for Raven DB) do not offer any transactions or atomic multi-document writes. Normally NoSQL databases lack security control such as foreign key, which is used by SQL databases for consistency control as well as validation of the database state. Other shortcomings of NoSQL include absence of stored procedures (in most of NoSQL Databases), and lack of triggers. Furthermore, while moving from SQL to NoSQL, a user must identify most fit NoSQL model for subject data, as direct mapping is not feasible. NoSQL databases support importing CSV dumps or JSON dumps, which makes the data migration easier as once the identification of data model is achieved only thing remains is exporting data to independent formats such as CSV. Such shortcomings make moving data SQL to NoSQL troublesome, hence the user must weigh all advantages and disadvantages before reaching to such a decision [25] [26] [27] [28].

iv) Poor support for transactional functions:

The two core design parameters which play primal part in NoSQL databases are i) faster key lookups and ii) operations which are atomic at the row level and not spanning the records. Due to these properties databases can be fragmented successfully (as there should never be spanning multiple machines in any kind of operation) and makes building scale out architectures easier. Characteristics like these makes the NoSQL more compatible to writing heavy workloads and faster rendering of webpages. Financial data is likely to be small in comparison to large web companies' data. Making scale out less important for finance sector. Moreover, financial data tends to be partitioned well because of which replica placement becomes more explicit [29] [30].

v) Compromised Clients

Clients accessing NoSQL databases may be in direct contact of resource managers or various nodes. Access from a single compromised location and malicious data can put the entire system at risk and compromise the security of data. In absence of central management security, protection of the clients, nodes and the name servers become tough [17].

Where relational databases offer more stability because of "seniority", they are a poor choice for large data or big data analysis. Contrastingly, NoSQL provides scalability, speed, schema-less approach, and support to big data applications along with risks and limitations as mentioned above.

This paper provides a broad overview on NoSQL databases risks and shortcomings; and tries to develop an audit checklist which can act as a basis for evolving research in the promising areas mentioned above and detailed research on the effects, usability, security, cost and comparison of different technologies used for security audit, can be undertaken to improve these technologies for the betterment of Information Security community as a whole.

III. SECURITY AUDIT

Database auditing includes observing the actions of database users and its characteristics to be aware. Auditing is frequently set up by database administrators and consultants for security reasons, for example, to ensure that those without the permission to access information do not access it. There are various types of audits which includes, Statement auditing, privilege auditing, schema object auditing, fine grained auditing and complete system auditing. Information audit comes under the systems audit category. Audit which is based and related to only the elements of Information System is called Information Audit. This fact makes the information audit excluded from the group of rest of the audits. The information audit's purpose is to give a process for recognizing, evaluating, and managing information resources to fully exploit information's strategic potential. The information audit should provide strategic direction and guidelines for the management of an organization's information resources considering its strategic purpose [31] [32] [33].

In this paper Information audit is used as a tool for identifying risks associated with NoSQL databases and their mitigations. Information audit is used as a first step towards development of more useful and less risk holding use of NoSQL databases. Information Audit has discreet procedures, which are used for achievement of certain objectives with specific tools. Processes, objectives, procedures, international regulations and components related to this are established by the global association for IT - ISACA (Information System Audit and Control Association) [32] [33].

Normally any type of databases and their strengths are measured with help of different frameworks such as ISO 38500, COBIT, ITIL, Calder-Moir IT Governance framework etc. These governance frameworks enable organizations to manage their IT risks effectively and ensure that the activities associated with information and technology are aligned with their overall

business objectives. Universal standards for IT governance and rules are called COBIT (Control Objectives for Information and Related Technology). In this paper COBIT framework is used for conducting audit of diverse NoSQL databases and measuring their impact on major IT framework characteristics. COBIT has defined seven important characteristics for any information framework: [32] [33]

- **Availability** – the information must be available at any time during the decision process.
- **Integrity** – the content and accuracy of the data must be in accordance with the rules and expectations of the organization.
- **Confidentiality** – the information must be provided only to users whom they are intended to be delivered.
- **Reliability** – the information must relate to the specific decision-making process that is served.
- **Efficiency** – the information must be provided with the lowest consumption of resources.
- **Effectiveness** – the information must be relevant, accurate and timely provided for decision making.
- **Compliance** – the logical structure of information and its concrete values must reflect the actual level of processes it characterizes.

These seven characteristics of information according to COBIT standard are important elements for the final analysis of IT audit. In this paper five (availability, integrity, confidentiality, reliability, and efficiency) of the seven characteristics are studied for three diverse NoSQL databases: Mongo DB, REDIS, and Couch DB. All risks and their impact on the NoSQL database are mapped with reference to these five characteristics [32] [33]. We have referred below versions of these databases to assess the security controls.

NoSQL Database	Version
MongoDB	4.4
Redis	6.2
CouchDB	3.1.1

Table 1: Referred Database versions

1) Security Audit for MongoDB

i) Check for default usernames and passwords

Consider a scenario where a vanilla instance of MongoDB has been installed and is to be configured the coming business days. In the meantime, an attacker gets access to the open ports of the database within the organization. For availability, accessing the instance will not directly affect the availability for the users of the database, but because of the motivation of the attacker. Confidentiality is immediately breached and the Reliability of the database/the group managing the database will be affected the most if the attacker can gain administrative access. Attackers could modify the entire database, if needed, which affects Integrity rating. Lastly, efficiency is not affected at all, because according to performance indicators attackers are using minimal resources within the database system.

ii) Verify encryption of data at rest:

For verifying data-at-rest encryption, it is assumed that the attacker has gained access to the enterprise/atlas database and not the master key (as it is stored in a different but a key management solution). Encryption of data at rest is not included in the free versions, but in the enterprise and the atlas versions.

Confidentiality, Availability and ultimately, Reliability are affected the least currently, as there are nil vulnerabilities found which could be exploited. Efficiency follows next, as due to heavy decryption of data, users can experience latency between 10%-20%. If the user has large amount of data writes/data, performance impacts can be high compared to read-only usage. After introduction of WiredTiger, performance impacts decreased more [Source]. For Integrity, if the attacker has read access and can perform a known-plaintext attack, or if the attacker has write access, attacker can modify the encrypted text into gibberish and corrupt user's data. Using non-repeated texts can thwart the known-plaintext attack, but the damage is more severe if user's data can be overwritten [34].

iii) Review database authorization and permissions granted to all users

Incorrect permission assignment can lead to nightmares for system administrators. Unless a vulnerability affecting permission model of Mongo DB is found, the reliability stands at maximum (0). Misconfigurations can lead to availability of non-essential processes for a user as well as not providing processes to a requesting user, which affects the availability characteristic. For the worst-case scenario of access to administrative processes given to a normal user, confidentiality of the data provided by the process is easily breached. Efficiency is affected when there are a large number of users requesting access to the database with a complex authorization model. Lastly, integrity depends on the modify/write access of the user, which can be of maximum impact if normal user has administrative rights.

iv) Security against quick attacks on open ports

Several systems are victims of unauthorized access where open ports are utilized. MongoDB has ports 27017 and 27018

in use for its internal processes, which can be scanned and if, for example, default or insecure credentials are utilized in the system, the characteristics might be heavily affected depending on the motivation of the attacker. Versions of MongoDB before 2.6.0 allowed remote unauthenticated connections, which lead to a spike in ransomware attacks for MongoDB, leading to the issue being rectified later [35] [36] [37] [38].

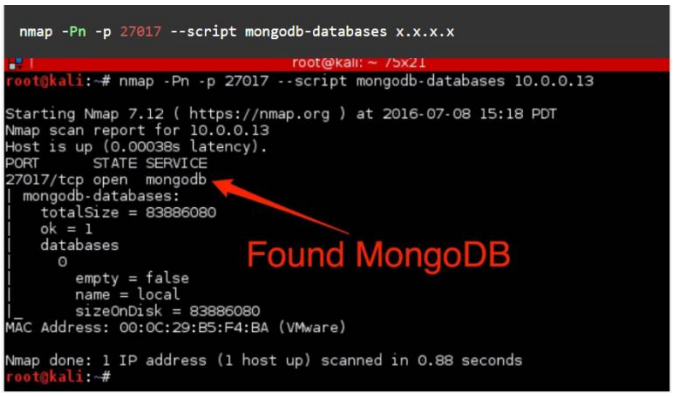

Fig. 1: Open Ports found in MongoDB

v) Security against DoS Attacks

Attackers using DoS attacks try to find vulnerabilities which disrupt the service availability for other users. There are several vulnerabilities discovered causing DoS attacks within the class CWE-400, including CVE-2015-4411, CVE-2016-3104, CVE-2014-8964 and CVE-2020-7926. These attacks are presented when multiple (in hundreds or thousands of) specially crafted queries are provided to the server, which then uses its resources to establish the query results, causing disruption of services for other legitimate users. For example, in CVE-2015-4411, while viewing the source code, characters '^' and '$' are used to start and terminate the Regular expression (instead of '\A' and '\z'), which causes the regex engine to validate the entire input string, even if the input is valid only up till a newline character [39] [40] [41] [42] [43].

`/^[0-9a-f]{24}$/` [44] validates the string till character 24 and approves even if there are invalid characters present thereon. Network filtering solutions such as firewalls and IPSs, strict access controls, dedicated user accounts for databases, and keeping updated versions of software are most common ways to avoid DoS attacks [44].

vi) Injection attacks

Injection attacks are steadily increasing within NoSQL paradigm. As with SQL injections, it is possible to gain access to a user account or administrative account if crafted queries are used. Various payloads are provided here: PayloadsAllTheThings. The documentation explicitly states that while utilizing several operations like mapReduce, $function, $where and $accumulator, using JavaScript can be an avenue for injection attacks.

Similar defense in depth strategies exist for protecting MongoDB against injection attacks, including but not limited to, frequent updates, least privileged access control, avoiding using Javascript at all if possible or using filters for vulnerable queries amongst others [45] [46].

2) ***Security Audit*** **for Redis**

i) Review weak password storage in Redis DB

Passwords are stored and transferred in plain text in Redis. The password is compromised if an attacker can listen in on the Redis server and the client. Similarly, if an attacker gains access to a Redis server's redis.conf configuration file, the password is stored in plain text and can be retrieved easily [47].

ii) Check for default username and passwords:

In Redis, the host is localhost, the default port is 6379, and there is no password by default if using a local instance [48].

Prior to Redis 6, Redis could only recognize AUTH password> is the one-argument form of the command. This form simply verifies the password entered with requirepass. If the password provided via AUTH matches the password in the configuration file, the server responds with the OK status code and begins receiving instructions. Otherwise, the client will receive an error message and will need to use a new password [48] [49].

Redis 6 has developed ACL commands where password is checked across the usernames using the command: AUTH <username> <password>. Then the command confirms the users and their passwords set in the redis.conf file to provide access to the server [49].

Because of Redis' fast performance, it is possible to test many passwords in a short period of time, thus, to prevent this attack make sure to create a strong and long password. The ACL GENPASS command is an effective technique to create strong passwords [49].

iii) Review database authorization and permissions granted to all users:

Password-based authentication or role-based access control are required for all Redis Cloud databases. One can define many users with fine-grained authorization features using role-based access control [50] [48].

To use role-based access control (RBAC), a Redis Cloud database supporting version 6.0.0 and above are needed [48].

The ACL command in open source Redis can be used to create users and assign Permissions to users. Open source Redis does not support generic roles [48].

If Redis is exposed, it is insecure, because Redis' security paradigm requires that only authorized and trustworthy clients have access to it. Unfortunately, this is insufficient. External clients can easily snoop into the Redis server and retrieve the desired data [50].

iv) Verify database as well as its users' authentication:

Redis does not seek to provide Access Control, but it does provide a minor layer of authentication that may be activated or removed via the redis.conf file. Unauthenticated clients will be denied access to Redis if the authorization layer is enabled. A client can authenticate themselves by using the AUTH command followed by the password [49].

The password is set in clear text in the redis.conf file by the system administrator. It needs to be strong enough to eliminate the chances of brute force attacks [49].

The authentication layer's goal is to create a redundancy layer that can be enabled or disabled. If firewalling or any other measure developed to protect Redis from external attackers fails, an external client will still be unable to access the Redis instance without knowing the authentication password [49].

Because the AUTH command, like all other Redis commands, is sent unencrypted, it is vulnerable to eavesdropping by an attacker with sufficient network access [49].

v) Verify encryption of data at rest:

Data on the network (data in motion) and data on disc (data at rest) are both encrypted by Redis Enterprise[51]. With a single checkbox, Redis Enterprise VPC also supports encryption at rest. Using transparent filesystem encryption capabilities available on Linux OS, administrators can encrypt data at rest in Redis Enterprise Software. The Enterprise version of Redis uses AWS Management console, that consists of ElastiCache navigation dashboard. In the cluster panel, if the attributes for encryption at rest and encryption in motion are unchecked by the administrator, the data is both stored and transmitted unencrypted causing the highest risk to the data. In the open-source version of Redis, the encryption of data is by default at both rest and in motion [49] [48].

vi) Review data migration from SQL to NoSQL:

Using ApsaraDB for Redis' pipeline feature, users can easily transfer data from ApsaraDB RDS for MySQL or on-premises MySQL databases to ApsaraDB for Redis[54].

ApsaraDB for Redis is used as a cache service between applications and databases to extend the capabilities of typical relational databases in one of the most common use cases. This benefits the ecosystem as well. Hot data is stored in ApsaraDB for Redis. ApsaraDB for Redis allows applications to immediately retrieve current data. To utilize ApsaraDB for Redis as a cache, users must first send data to ApsaraDB for Redis from a relational database. Tables from a relational database cannot be immediately transferred to the ApsaraDB for Redis database, which stores data in a key-value structure [50].

vii) Ensure/Verify security against quick attacks on open ports:

Allowing the open ports on internet to access a misconfigured service could allow an attacker to gain a foothold on the server, in addition to exposing the data. Attacks on Redis have been carried out that add an attacker's SSH key to the authorized keys file of the Redis user, enabling the attacker SSH access [48] [47] [49].

viii) IP Binding Security

Redis, by default is configured to be accessed by all addresses (#bind 127.0.0.1). Multiple addresses con be configured using command # bind 127.0.0.1 newip. Configuring multiple addresses means that this address can be accessed by other applications, not the address of the application service. For instance, Application A, B, C, D, is deployed in the intranet environment, they need to use Redis service (The application service and Redis service are in the same intranet). When the Redis service starts, it will use the internal network address where the own server is located that gets exposed [48].

Application A, B, C, D, can be used to access the Redis service, if these applications are not configured, an exception will be given, "**Could not connect to Redis at 192.1.17.61:6379: Connection refused**" [48].

After the setting, access can be control by allowing only the machines on the intranet can access the Redis service. but, if any other machine in the intranet is invaded, then the Redis server is also insecure. It can be made secure by adding a password to the Redis server. Password should be set/modified in redis.conf file and then the machine should be restarted [48] [49].

ix) Ensure/Verify security against DOS attacks:

An attacker can launch a certain type of attack from the outside, even if they do not have access to the instance. The ability to input data into Redis that causes pathological (worst case) algorithm complexity on data structures defined inside Redis internals is an example of such an attack [50].

For example, an attacker could send a series of strings known to hash to the same bucket into a hash table using a web form to convert the O(1) expected time (average time) to the O(N) worst case, consuming more CPU than expected and resulting in a Denial of Service [50].

Redis uses a per-execution pseudo-random seed to the hash function to prevent this specific attack [50].

The qsort algorithm is used by Redis to implement the SORT command. Because the algorithm is currently not randomized, a quadratic worst-case behavior can be induced by carefully picking the proper set of inputs [50].

x) Verify security against Injection attacks:

Because the Redis protocol has no conception of string escaping, injection using a standard client library is difficult in most cases. The protocol is binary safe and use prefixed-length strings [50].

3) ***Security Audit of*** **CouchDB**

i) Review weak password storage in NoSQL DB

CouchDB does not save the plain-text password anywhere, it rewrites the plain-text passwords using a hashing algorithm, so they are hashed [51]. Whenever a user is created, password gets hashed right away in local.ini file [51]. Even if the stored hash enters the hands of an attacker, recovering the plain-text password from the hash is currently too inconvenient i.e., it will cost a lot of money and effort.

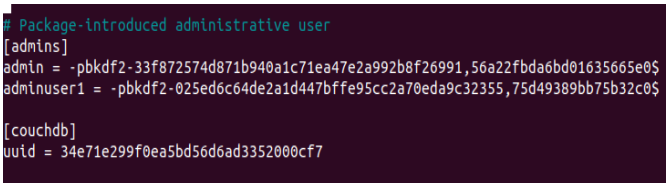

Fig. 2: Hashed password in CouchDB

ii) Check for default username and passwords

An admin user (e.g., an administrator, a super user, or root) in CouchDB can do anything with the database. Everyone is an administrator by default. Additionally, individual admin users with a username and password can be created [51]. CouchDB can be compromised with Brute Force attack with default username admin and with an easily guessable password, as the number of login attempts is not limited. To implement standard password management, it is recommended to follow below procedures:

- While creating user account the strength of password should be estimated to enforce strong passwords.
- Incorrect login attempts must be limited to 3 or 4 and account should be locked after that.
- Inactive accounts after a certain period must be deactivated automatically.

iii) Review database authorization and permissions granted to all users

Each user in CouchDB has been categorized into either admin or a member. Database's protection in this is determined by the security object, which is a JSON document made up of two components: "admins" and "members," each of which is an object with two more components: "names" and "roles," the values of which are arrays [51].

The ability to build design documents (view definitions and other specific formulas, as well as standard fields and blobs, are all contained in these special documents) and alter a database protection object is added to such permissions by a named list of privileges called "admins" [51]. “admin” and “adminuser1” were created as admins.

Permissions to build, edit, read, and delete documents in the database are granted by a named collection of privileges known as "members." “mem1” and “mem2” users were created as members.

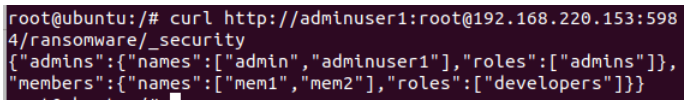

Fig. 3: Users created in the CouchDB

iv) Verify database as well as its users' authentication

CouchDB administrators are declared inside the config file whereas the regular users are stored in a separate authenticated database also known as System Database. This database contains all the registered users as JSON documents and is named as ‘_users’ by default. Unlike other databases only administrators can access the documents in the System Database, can do changes and can execute design functions. It is verified that the regular users can access and do changes only to the documents they have created.

v) Verify encryption of data at rest (Assumption: Access is violated)

The ability to encrypt a database (partition) on disk is important in some highly sensitive application domains. In CouchDB, database encryption is not supported [52].

vi) Review data migration from SQL to NoSQL

SQL Server and CouchDB are extremely different technologies. There is no one-to-one migration path available because neither SQL nor CouchDB has similar features of the other. It is not so easy to migrate data from MySQL to CouchDB because MySQL is a relational database that has been normalized to at a minimum third normal form and preferably further, the other CouchDB is a document based datastore that is not relational and is not normalized. So, there is no natural correspondence between elements in either that could apply a generalized algorithm to perform an export. One possible way is to map the schema of the MySQL to the document of CouchDB and write an SQL query to export to text and then can be imported to CouchDB [51]. There are few third-party tools available to migrate data automatically from SQL server to CouchDB, it converts the data in tabular form to JSON Format that would fit to CouchDB systems schema [53].

vii) Check IP Binding Security

To make any requests to CouchDB, by default it listens only to local host or loopback IP address i.e., 127.0.0.1. It will be open to public when the system public IP address is bind to the database, but the database access will be restricted only to admins (admin credentials must be included along with the public IP address) [51].

viii) Ensure/Verify security against quick attacks on open ports

Latest version of CouchDB by default listens to port number 5984 for http and 6984 for https. Any other random ports that are open and listening to all the interfaces can be Erlang VM’s distribution port that is used in clustered mode of CouchDB to connect to another CouchDB node to form a cluster. Port Number 4369 is usually used as Erlang Port mapper daemon (epmd) which plays major role to achieve cluster installation. It is used to find other CouchDB nodes, so all servers can communicate to each other on this port. Risk on exposing this port to internet or any untrusted network can be protected by the Erlang cookie – monster in this NoSQL database [54] [54].

ix) Ensure/Verify security against DOS attacks

CouchDB is vulnerable to Denial-of-Service (DOS) attacks [55]. All networked servers are subject to denial-of-service attacks. Such attacks can be reported to the Apache Software

Foundation through a private security mailing list to address the issue [51].

x) Verify security against Injection attacks

CouchDB is vulnerable to script injection. CouchDB enables the ability to run JavaScript in the database engine to conduct complex queries or transactions, such as map reduce, is a typical feature of NoSQL databases. If un-escaped or inadequately escaped user input makes its way to the query, JavaScript execution exposes a severe attack surface [56].

The authors consider the previous marking balanced score table as an inappropriate system for the current research due to various reasons. The previous paper focuses on building a tool for streamlining output of various audit software whereas this focuses on building a checklist-based framework on risks/threats of NoSQL flavors, which is a major difference in the implementation of the score table for Databases. The previous research considers SQL databases and not NoSQL, whereas this research considers the latter. Furthermore, the measurement guide does not specify a baseline score or a baseline implementation of related applications for guiding an organization to fill the characteristic table, making the values for the audit checklist to be relative to the individual's experience, opinions, and bias. Moreover, the research doesn't consider the limitations of the company as to how a score is obtained. Mistakes made by inexperienced or veteran developers or bugs introduced by implementing patches can introduce several variants of vulnerabilities. These vulnerabilities can then be exploited by using various TTPs. These scenarios are partially examined in the current study according to the authors' best knowledge. Lastly, this research provides recommendations or plausible solutions, wherever applicable.

Grading Scale used:

Weak - The attacker can impact the data characteristic in a severe manner. For example, using default administrative credentials can lead to severe consequences on confidentiality, reliability and availability of the database.
Medium - The attacker can achieve breach of data characteristic with high effects in some scenarios while low in other, depending on the access they have, motivations/intent and capabilities. For example, using default credentials to gain access in a database can lead to little to no availability problems, unless there is a restriction of one-user-X-machine access, or the attacker introduces a database encryptor and/or removes database access for the user.
Strong - There is a low effect of attack on the data characteristic in the researched scenarios. For example, for most of described vulnerabilities, only the attacker and other users access the database simultaneously, similar to any other scenario where same number of legitimate users with the compromised user access the database.

The research considers only some situations, and it is not exhaustive.

NoSQL Security Controls	Availability of the controls in the NoSQL DBMS's		
	MongoDB	**Redis**	**CouchDB**
Review weak password storage in NoSQL DB	Weak	Weak	Strong
Check for default username and passwords	Weak	Strong	Weak
Review database authorization and permissions granted to all users	Weak	Weak	Weak
Verify database as well as its users' authentication	Not Assessed	Weak	Weak
Verify encryption of data at rest	Weak	Strong	NA
Review data migration from SQL to NoSQL	Medium	Medium	Weak
Check IP binding security	Medium	Medium	Weak
Ensure/Verify security against quick attacks on open ports	Weak	Weak	Medium
Ensure/Verify security against DOS attacks	Weak	Medium	Weak
Verify security against Injection attacks	Weak	Not Assessed	Weak
Check support for transactional functions in NoSQL database	Not Assessed	Not Assessed	Not Assessed

Table 2: Availability of security controls in NoSQL DBMS

IV. DISCUSSION

This research has been performed based on the specified versions of Redis, CouchDB and MongoDB. This paper shows the audit of various security controls that could impact the performance of the NoSQL databases. The table in the section III. illustrates that with the default settings, the NoSQL databases are vulnerable and do not provide any security to the users and their data storage.

However, some DB's provide plugins that can enhance the security of the DB, for instance, Redis provides 'protected mode' to secure the connections to its server from external devices, but this mode can also be stopped by using stop command. Likewise, the other NoSQL DB could also be snooped in by different types of attacks.

This research could help the developers, users, auditors, and the researchers of NoSQL DBMS to focus on the various default controls that are vulnerable to attacks and try to verify all the controls to secure the datasets and improve security aspect along with the high-performance.

V. CONCLUSION

Security audit becomes of primary importance for an organization possessing substantial (and sometimes valuable) amount of data. This paper approaches the auditing problem from a perspective of risk-based NoSQL audit, by identifying several security risks associated with NoSQL databases and their possible controls. Three database types were considered: MongoDB, CouchDB and Redis for their adherence to various security controls based on commonly faced risks and conjecture, and some recommendations are suggested for the associated risks. Furthermore, this paper focuses on providing recommendations based on risks and vulnerabilities specific to NoSQL databases, for which very few studies exist, implying for its usage as a baseline or a guideline for future research. Few of the derived security controls prospectively can become candidates that determine the audit standard of the default NoSQL databases.

Proposed security audit can aid businesses implementing NoSQL as a part of their security stack on various operational levels and can also be used by database developers and administrators in identifying security concerns connected with NoSQL implementations.

VI. References

[1] Akingbade, "Cloud Storage problems, benefits and solutions provided by Data," *International Journal of Engineering and Innovative Technology*, vol. 5, pp. 70-77, 2016.

[2] G. R. Gema Rodríguez-Pérez, A. Serebrenik, A. Z. M. Germán and J. Gonzalez-Barahona, "How bugs are born: a model to identify how bugs are introduced in software components," 4 Feb 2020. [Online]. Available: https://link.springer.com/article/10.1007/s10664-019-09781-y. [Accessed May 2021].

[3] Lineup Systems | Industry Analysis, "3 Reasons You Should Audit Your Database Regularly," Lineup, 24 March 2020. [Online]. Available: https://www.lineup.com/newsroom/industry-analysis/database-audit. [Accessed May 2021].

[4] DataSunrise, "Why You Need a Database Audit Trail," DataSunrise, [Online]. Available: https://www.datasunrise.com/blog/professional-info/why-you-need-a-database-audit-trail/. [Accessed May 2021].

[5] D. Bisson, "7 Data Breaches Caused by Human Error: Did Encryption Play a Role?," Venafi, 15 October 2020. [Online]. Available: https://www.venafi.com/blog/7-data-breaches-caused-human-error-did-encryption-play-role. [Accessed May 2021].

[6] C. Bradford, "7 Most Infamous Cloud Security Breaches," StorageCraft: an Arcserve Company, [Online]. Available: https://blog.storagecraft.com/7-infamous-cloud-security-breaches/. [Accessed May 2021].

[7] M. Korolov, "Cloud security configuration errors put data at risk; new tools can help," CSO, 29 Jan 2018. [Online]. Available: https://www.csoonline.com/article/3251605/cloud-security-configuration-errors-put-data-at-risk-new-tools-can-help.html. [Accessed May 2021].

[8] H. Kaur, "SQL vs NoSQL: Which one is better to use?," September 2020. [Online]. Available: https://www.geeksforgeeks.org/sql-vs-nosql-which-one-is-better-to-use/.

[9] C. Tozzi, "The Limitations of NoSQL Database Storage: Why NoSQL's Not Perfect," May 2016. [Online]. Available: https://www.channelfutures.com/cloud-2/the-limitations-of-nosql-database-storage-why-nosqls-not-perfect.

[10] A. Fowler, "10 Advantages of NoSQL over RDBMS," dummies, [Online]. Available: //www.dummies.com/programming/big-data/10-advantages-of-nosql-over-rdbms/.. [Accessed March 2021].

[11] "Advantages of NoSQL Databases," mongoDB, [Online]. Available: https://www.mongodb.com/nosql-explained/advantages. [Accessed March 2021].

[12] U. Saxena and S. Sachdeva, "An Insightful View on Security and Performance of NoSQL Databases," in *International Conference on Recent Developments in Science, Engineering and Technology*, Springer, Singapore, 2017.

[13] Aarti, "Advantages and Disadvantages of NoSql:-," 27 May 2020. [Online]. Available: https://aartiy734.medium.com/advantages-and-disadvantages-of-nosql-2b285ba64b96. [Accessed Jan 2021].

[14] H. Mackin, G. Perez and C. C. Tappert, "Adopting NoSQL Databases Using a Quality Attribute Framework and Risks Analysis," in *y SCITEPRESS – Science and Technology Publications, Lda*, New York, 2016.

[15] L. Olivera, "Everything you need to know about NoSQL databases," DEV, 4 Jun 2019. [Online]. Available: https://dev.to/lmolivera/everything-you-need-to-know-about-nosql-databases-3o3h#dis. [Accessed March 2021].

[16] Microsoft, "Security Monitoring and Attack Detection," 5 November 2014. [Online]. Available: https://docs.microsoft.com/en-us/previous-versions/tn-archive/cc875806(v=technet.10)?redirectedfrom=MSDN.

[17] K. Ahmad, M. S. Alam and N. I. Udzir, "Security of NoSQL Database Against Intruders," *Recent Patents on Engineering*, 21 June 2018.

[18] T. Karavasilev and E. Somova, "Overcoming the Security Issues of NoSQL Databases," © *Journal of the Technical University - Sofia*, vol. 24, no. "Fundamental Sciences and Applications", May 2018.

[19] E. Sahafizadeh and M. A. Nematbakhsh, "A Survey on Security Issues in Big Data and NoSQL," *Advances in Computer Science : an International Journal*, vol. 4, no. 4, p. 5, 2015.

[20] A. Zahid, R. Masood and A. Shibli, "Security of sharded NoSQL databases: A comparative analysis," in *Conference on Information Assurance and Cyber Security (CIACS)*, Rawalpindi, Pakistan, 2014.

[21] Apache Software Foundation, "Apache CouchDB," Feb 2021. [Online]. Available: https://docs.couchdb.org/en/latest/api/server/authn.html#authentication.

[22] P. Aggarwal and R. Rani, "Security Issues and User Authentication in MongoDB," in *Elsevier*, Patiala, India, 2014.

[23] "Store Passwords In MongoDB With Node.js, Mongoose, & Bcrypt," Coder Rocket Fuel, 30 Aug 2020. [Online]. Available: https://coderrocketfuel.com/article/store-passwords-in-mongodb-with-node-js-mongoose-and-bcrypt. [Accessed March 2021].

[24] K. Magee, "IT Auditing and Controls – Database Technology and Controls," INFOSEC, 2 July 2011. [Online]. Available: https://resources.infosecinstitute.com/certification/itac-database/. [Accessed March 2021].

[25] J. Himango, "DZone > Database Zone > The Biggest Challenges of Moving to NoSQL," 20 Sep 2017. [Online]. Available: https://dzone.com/articles/the-biggest-challenges-of-moving-to-nosql. [Accessed Feb 2021].

[26] J. Lewkowicz, "The move from RDBMS to NoSQL requires optimizing over time," SDTimes, 19 Nov 2020. [Online]. Available: https://sdtimes.com/data/the-move-from-rdbms-to-nosql-requires-optimizing-over-time/. [Accessed March 2021].

[27] F. Oliveira, A. Oliveira and B. Alturas, "Migration of Relational Databases to NoSQL - Methods of Analysis," in *7th International Conference on Humanand Social Sciences*, Barcelona: Richmann Publishing, 2018.

[28] C. Vithalani, "Project 4 - SQL to NoSQL migration," 2016.

[29] S. Severance, "StackExchange: Quantitative Finance," Dec 2011. [Online]. Available: https://quant.stackexchange.com/questions/1392/usage-of-nosql-storage-in-finance. [Accessed Jan 2021].

[30] C. Tozzi, "The Limitations of NoSQL Database Storage: Why NoSQL's Not Perfect," Channel Futures, 31 May 2016. [Online]. Available: https://www.channelfutures.com/cloud-2/the-limitations-of-nosql-database-storage-why-nosqls-not-perfect. [Accessed March 2021].

[31] I. RUS, "Technologies And Methods For Auditing Databases," *Procedia: Economics and Finance*, pp. 991-999, 2015.

[32] "Control Objective for IT-related Technology (CobIT)," Hendershett Consulting Inc. , [Online]. Available: https://hci-itil.com/COBIT/cobit_overview.html. [Accessed May 2021].

[33] COBIT Steering Committee and the IT Governance Institute, "COBIT Framework 3rd Edition," July 2000. [Online]. Available: http://www.nekodasuke.jp/cism/COBIT/COBIT3_3_framewrk.pdf. [Accessed May 2021].

[34] Townsend Security, "The definitive guide to Mongodb encryption & key management," Townsend Security, [Online]. Available: https://info.townsendsecurity.com/mongodb-encryption-key-management-definitive-guide. [Accessed May 2021].

[35] T. Kadlec, "The MongoDB hack and the importance of secure defaults," snyk, 10 Jan 2017. [Online]. Available: https://snyk.io/blog/mongodb-hack-and-secure-defaults/. [Accessed May 2021].

[36] D. Riley, "Ransomware targeting MongoDB databases threatens to report victims for GDPR breach," SiliconAngle, 2 Jul 2020. [Online]. Available: https://siliconangle.com/2020/07/02/ransomware-targeting-mongodb-databases-threatens-report-victims-gdpr-breach/. [Accessed May 2021].

[37] D. Ottenheimer, "Update: How to Avoid a Malicious Attack That Ransoms Your Data," MongoDB Documentation, 7 Sep 2017. [Online]. Available: https://www.mongodb.com/blog/post/update-how-to-avoid-a-malicious-attack-that-ransoms-your-data. [Accessed May 2021].

[38] B. Brenner, "Thousands of MongoDB databases compromised and held to ransom," NakedSecurity, 11 Jan 2017. [Online]. Available: https://nakedsecurity.sophos.com/2017/01/11/thousands-of-mongodb-databases-compromised-and-held-to-ransom/. [Accessed May 2021].

[39] CVE, "CVE-2020-7926," CVE Org., [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-7926. [Accessed May 2021].

[40] CVE List, "CVE-2014-8964," CVE Org, [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-8964. [Accessed May 2021].

[41] CVE List, "CVE-2016-3104," CVE Org., [Online]. Available: https://www.cvedetails.com/cve/CVE-2016-3104/. [Accessed May 2021].

[42] CVE List, "CVE-2015-4411," CVE Org, [Online]. Available: https://www.cvedetails.com/cve/CVE-2015-4411/. [Accessed May 2021].

[43] CWE Definitions, "CWE-400: Uncontrolled Resource Consumption," CWE: Common Weakness Enumeration, [Online]. Available: https://cwe.mitre.org/data/definitions/400.html. [Accessed May 2021].

[44] "mongodb: Comparing changes," GitHub, [Online]. Available: https://github.com/mongodb/bson-ruby/compare/7446d7c6764dfda8dc4480ce16d5c023e74be5ca...28f34978a85b689a4480b4d343389bf4886522e7. [Accessed May 2021].

[45] L. Johnson, "Presenting in the Real World: Bananas with MongoDB," Rapid7, 28 Jul 2016. [Online]. Available: rapid7.com/blog/post/2016/07/28/pentesting-in-the-

real-world-going-bananas-with-mongodb/. [Accessed May 2021].

[46] "PayloadsAllTheThings/NoSQL Injection/," GitHub, [Online]. Available: https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/NoSQL%20Injection. [Accessed May 2021].

[47] J. Nelson, Mastering Redis, Mumbai: Packt Publishing, 2016.

[48] Redislabs, "Redis," [Online]. Available: https://redis.io/topics/security.

[49] Redislabs, "Redis," 2015. [Online]. Available: https://redis.io/commands/auth.

[50] B. Cihan, "Securing Redis with Redis Enterprise for Compliance Requirements," Jan 2018. [Online]. Available: https://redislabs.com/blog/securing-redis-with-redis-enterprise-for-compliance-requirements/.

[51] Apache CouchDB, 2020. [Online]. Available: https://docs.couchdb.org.

[52] QuABaseBD, "CouchDB Security Features," Feb 2015. [Online]. Available: https://quabase.sei.cmu.edu/mediawiki/index.php/CouchDB_Security_Features.

[53] Safe Sotware, "Migrate Data from SQL Server to CouchDB," 2021. [Online]. Available: https://www.safe.com/convert/sql-server/couchdb/.

[54] Apache CouchDB, 2021. [Online]. Available: https://docs.couchdb.org/en/latest/setup/cluster.html.

[55] S. Gadhiraju , "Assessing the vulnerabilities and securing MongoDB and Cassandra databases," Culminating Projects in Information Assurance, 2020. [Online]. Available: https://repository.stcloudstate.edu/msia_etds/107.

[56] A. Ron, A. Shulman-Peleg and E. Bronshtein, "No SQL, No Injection? Examining NoSQL Security," 2015.