Professor Elvira NICA, PhD
E-mail: elvira.nica@ase.ro
The Bucharest University of Economic Studies
Senior Lecturer Bogdan George TUDORICA, PhD
E-mail: bogdan.tudorica@upg-ploiesti.ro
Petroleum-Gas University of Ploiesti
Professor Dorel-Mihail DUSMANESCU, PhD
E-mail: doreld@upg-ploiesti.ro
Petroleum-Gas University of Ploiesti
Professor Gheorghe POPESCU, PhD
E-mail: popescu_ucdc@yahoo.com
Dimitrie Cantemir Christian University
Lecturer Alina Maria BREAZ, PhD
E-mail: alinamariabreaz@gmail.com
Aurel Vlaicu University of Arad

DATABASES SECURITY ISSUES - A SHORT ANALYSIS ON THE EMERGENT SECURITY PROBLEMS GENERATED BY NoSQL DATABASES

Abstract. *Over the last few years, the NoSQL (Not only SQL) databases managed to impose themselves as a suitable alternative to their more common relational counterparts. As such, the NoSQL databases have a plethora of practical applications, such as various Web 2.0 solutions, document management but also real-time systems where low latency is critical, embedded systems, storage for sensor networks and finally the IoT (Internet of Things). There are enough differences between the NoSQL and relational databases to presume that the security problems related to each of the two types of databases are at least partially different. This paper tries to review a few of the security aspects associated with the NoSQL databases usage.*

Keywords: *NoSQL; security; threats; vulnerabilities.*

JEL Classification: C88, O31, O33

1. Introduction

The central idea to the NoSQL approach is the fact that the classic relational databases, although representing a very mature and robust technology, have several limitations: volume of data, horizontal scalability, performance in write operations,

DOI: 10.24818/18423264/53.3.19.07

fast access, flexibility, complex maintainability, administration and operations and so on.

To solve all these limitations, various NoSQL databases use several different architectural approaches(key-value store, document store, wide column store, etc.).

Building on this, the new type of databases has some advantages (they are faster than their predecessors both in the basic read / write operations and in the data extraction operations, they are more apt to store very large amounts of data, they are more flexible) but also some disadvantages (normalization and ACID - Atomicity, Consistency, Isolation, Durability - are most of the times not available, the query methods or equivalents are not that capable of complex operations – e.g. joins)(Edlich, 2012).

It should be noted that there are some NoSQL solutions which do include ACID capabilities (MarkLogic, Aerospike, FairCom c-treeACE, Symas LMDB and OrientDB) and there are even solutions which do offer both ACID and join capabilities (Google Spanner, Clustrix, VoltDB, MemSQL, Pivotal's GemFire XD, SAP HANA, NuoDB, and Trafodionare) but these ones are more often classified as NewSQL instead of NoSQL.

Even though we did not commented anything to this point on the architectural intricacies of the various NoSQL solutions, we should note that, not only the NoSQL databases are vastly differing from the relational ones, but they are even greatly differing from each other – DB-Engines is listing 121 relational solutions in 1 category and 184 solutions, which can be labeled lato sensu as NoSQL, in 11 categories(Anon., 2016). As a direct consequence, one should not expect to find a single set of security related issues with the NoSQL solutions (as was the case with the relational databases), but multiple sets (maybe one related which each subtype of NoSQL databases).

Talking about expectancy, what should we be really presuming? In our opinion, we should be expecting two broad categories of NoSQL security related issues:

- Known security vulnerabilities adapted to the new environment;
- Brand new security vulnerabilities generated by the new methods and technologies used in the NoSQL solutions.

Note. The following review is mostly related to the main subtypes of NoSQL databases (key-value store, document store, wide column store). The other subtypes of NoSQL Database Management Systems (Graph DBMS, Time Series DBMS, Search Engine DBMS, Content stores, Multi-value DBMS, Native XML DBMS, Object Oriented DBMS, and RDF Stores) are only scarcely covered in the existing security related literature.

Note. Threats not directly related with the NoSQL solutions (e.g. DDoS type attacks) will not be covered in this paper.

DOI: 10.24818/18423264/53.3.19.07

2. Known security vulnerabilities

When we talk about known security vulnerabilities, we should look no further than The Open Web Application Security Project (OWASP) Top 10 Application Security Risks – 2013 list(The OWASP Foundation, 2016). We can also add to that the CWE/SANS Top 25 Most Dangerous Software Errors list(Martin, et al., 2011) and the CWE/SANS On the Cusp: Other Weaknesses to Consider list(Martin, et al., 2011).

OWASP Top Ten is not an exhaustive list - more can be found in the OWASP Developer Guide, OWASP Testing Guide and OWASP Code Review Guide - but it covers the main vulnerabilities which can be found in online applications (we should not forget that the vast majority of applications which are using NoSQL applications are web application or have at least a web component). Let’s make a quick review of these vulnerabilities.

A. Injection

There are multiple types of injection which can occur (e.g. SQL, OS, and LDAP injection are classical injection types). The injection vulnerability appears when untrusted data can be sent to an interpreter as part of a command or query. This way the interpreter can be put to execute unintended commands or to access data without proper authorization(The OWASP Foundation, 2016)(Kalman, 2014).

For NoSQL databases, SQL injection seems to be out of the scene while OS and LDAP injection are still feasible. Of course, the SQL itself cannot be used, but for each NoSQL solution there are still ways to query the database –it’s a basic functionality for a database.

Java Script Object Notation (JSON) is a new way of querying databases (especially NoSQL databases). The following DBMS are using JSON: Druid, Elasticsearch, CouchDB, RavenDB, MarkLogic Server, JSON Object Document Mapper, JasDB, RaptorDB, Embedded JSON DB, SDB, Riak, Scalien, Pincaster, RaptorDB, InfoGrid, ArangoDB, MarcelloDB, Axibase, VaultDB and MongoDB(Edlich, 2012).

As a querying method, JSON also becomes an attack vector for injection attacks. Several JSON injection examples are provided by (Ron, et al., 2015), (Oku, 2014),(Sullivan, 2011).

A JSON injection can take the form of a direct command JSON injection (which can be built on a Broken Authentication and Session Management vulnerability), a PHP array JSON injection, a HTTP POST JSON injection (maybe exploited further via a Cross-Site Request Forgery), or a JavaScript JSON injection (a classical Cross-Site Scripting vulnerability)(Ron, et al., 2015).

DOI: 10.24818/18423264/53.3.19.07

Other forms of query related injection (e.g. CQL injection) can affect Cassandra, NEO4J, Hadoop/HBASE and other NoSQL DBMS(Kadebu & Mapanga, 2014).

B. Broken Authentication and Session Management

Authentication and session management are main application functionalities. Often they are not implemented correctly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities(The OWASP Foundation, 2016)(Kalman, 2014).

Multiple NoSQL solutions (e.g. MongoDB, Redis, CouchDB, Cassandra, NEO4J and Hadoop) seem to be suffering from such vulnerabilities for they are presumably executed in trusted environments(Kadebu & Mapanga, 2014)(Chow, 2013)(Yegulalp, 2015).

This fact, the lack of authentication measures from the default deployments of some NoSQL DBMS, was actually perceived for a while as a feature by several vendors. A few examples of this are(Sullivan, 2011):

- "One valid way to run the Mongo database is in a trusted environment, with no security and authentication" … This "is the default option and is recommended" - the MongoDB documentation
- "The default AllowAllAuthenticator approach is essentially pass-through" - the Cassandra Wiki
- The "Admin Party": Everyone can do everything by default - CouchDB: The Definitive Guide
- No authentication or authorization support - Riak

For another example, MongoDB, which is ranked as the most popular NoSQL solution by DB Engines (DB Engines, 2016), was analyzed in multiple occasions during 2014 and found to have a huge number of installations not secured (Rossi, 2015)(Butturini, 2014) (56.5% from the total of MongoDB installations found in one of the studies amounting for 18000+ vulnerable installations, 40000+ vulnerable installations in another study).

At later moments some of the NoSQL vendors started to take care of these vulnerabilities (the work is in progress at various NoSQL solutions). As an example, MongoDB started to implement valuable security features in version 2.6 (now at version 3.2).At this moment security checklists for this product are available and a security architecture is made public and can be put in place.

Note. One of the NoSQL products evolution characteristics is the fact that new major versions are often radically different than the previous ones, not maintaining backward compatibility (or declaring most of the previously used commands and methods as obsolete and doubling them with new ones while the old ones are still available for a while).

DOI: 10.24818/18423264/53.3.19.07

C. Cross-Site Scripting (XSS)

XSS attacks can be made when an application takes untrusted data and sends it to a web browser without proper validation or escaping. XSS allows attackers to execute scripts in the target's browser and thosescripts can hijack user sessions, deface web sites, or redirect the user to malicious sites(The OWASP Foundation, 2016)(Kalman, 2014).

Many NoSQL solutions are using JavaScript as client-side and / or server-side scripting language: MongoDB, Druid, CouchDB, JSON ODM, NeDB, NoSQL embedded DB, KitaroDB, ArangoDB, gunDB, eXist, IBM Lotus/Domino(Edlich, 2012).

As it was already seen in the Injection section, JavaScript can be used as an auxiliary for various injection attacks. Bryan Sullivan provides multiple forms of JavaScript enabled attacks in (Sullivan, 2011). Some other JavaScript related vulnerabilities are also mentioned in (Ron, et al., 2015) and (Chow, 2013).

D. Insecure Direct Object References

An insecure direct object reference happens when the developer exposes a reference to an internal implementation object, be it the entire database, a file, a folder, or a database key. Without a way to control the access, these references can be manipulated to access unauthorized data(The OWASP Foundation, 2016)(Kalman, 2014).

Many of the existing NoSQL solutions have this vulnerability for they are using default port numbers which are open. Taking into account that most of the times the administrators / developers are not changing these ports and it's not very difficult to find the database vendor and the IP address, accessing the database becomes very easy(Chow, 2013):

- MongoDB: 27017, 28017, 27080
- CouchDB: 5948
- Hbase: 9000
- Cassandra: 9160
- NEO4J: 7474
- Redis: 6379
- Riak: 8098

E. Security Misconfiguration

Beyond other security controls, a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform is a must for ensuring a good level of security(The OWASP Foundation, 2016)(Kalman, 2014). A common trait of many NoSQL DBMS is the fact that they

DOI: 10.24818/18423264/53.3.19.07

have insecure configurations by default. It is the work of a good DB administrator to change a default insecure configuration into a working secure one.

Additionally, software should be kept up to date, a task which is, again, the signature of a good DB administrator. Unhappily, one of the things badly lacking in the NoSQL area is good DB administrators (we can actually say that, in this area, not only good administrators, but administrators, generally speaking, are absent)(Singh, 2016).

F. Sensitive Data Exposure

Many of the NoSQL solution, while still being in their infancy stages, are lacking protection by encryption for data at rest (in storage) and / or in transit. Also deficient are special precautions when the data are exchanged with the browser(Yegulalp, 2015)(Singh, 2016)(Kirkpatrick, 2013).

A particularly severe issue is the lack of encryption or weak encryption for very sensitive data such as password storage. (Chow, 2013) reported that at the respective moment (2013) MongoDB was using MD5 for password storage, Redis was storing passwords as plain text and CouchDB was storing passwords as plain text or encrypted them using weak salts.

Note. Later developments of the most preeminent NoSQL solutions introduced several encryption and auditing features (such as encryption at rest and encryption in transit in MongoDB, version 2.6, auditing in version 2.6(Butturini, 2014) and full support for SSL in later versions – 3.0 and 3.2).

G. Missing Function Level Access Control

Web applications are supposed to check for the right level of access both when a feature is made accessible in the UI and on the server at the moment each function is accessed. Some of the applications are making all the necessary verifications and this is a fact for both applications built on relational DB and for applications built on NoSQL DB. One would think that the level of vulnerability should be the same but this is not true. The same idea expressed earlier (the lack of experience) about the NoSQL DB administrators is also true about NoSQL developers.

Note. Taking into account the five years adoption rule of thumb and stating that a true beginning of NoSQL usage is somewhere at the beginning of 2013 (as it can be deduced from(Anon., 2016); we know that the real beginnings of the NoSQL phenomenon are dated way earlier, but we are talking about the relative moment when the NoSQL solutions became real life / commercial products), one should think that the beginning of the maturity stage in the NoSQL world should occur no earlier than late 2017- early 2018.

DOI: 10.24818/18423264/53.3.19.07

H. Cross-Site Request Forgery (CSRF)

A CSRF attack is based on the idea of using a logged-on victim's browser to send a forged HTTP request, including the information from the legitimate session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to use the victim's browser to generate requests which are considered genuine by the application / server. As a response to these requests, the application can return unauthorized data or can execute various data manipulation tasks(The OWASP Foundation, 2016)(Kalman, 2014).

For NoSQL based applications, this vulnerability applies as well as for relational ones, for many NoSQL solutions are providing various flavors of HTTP / REST API. Amongst these we can find: MongoDB, CouchDB, Hbase, Druid, RavenDB, MarkLogic Server, Clusterpoint Server, Terrastore, BangDB, Scalien, Pincaster, TreodeDB, TITAN, InfoGrid, BrightstarDB, ArangoDB, Create Data, eXist, Qizx, GT.M, OpenInsight, Model 204 Database, IBM Lotus/Domino(Edlich, 2012)(Ron, et al., 2015).Several REST based attacks are described in (Ron, et al., 2015) and (Sullivan, 2011).

I. Using Known Vulnerable Components

Usually software components, such as libraries, frameworks, and other software modules run with full privileges. As such, the exploitation of a known vulnerability in a software component means accessing data with full privileges, including a possible server takeover(The OWASP Foundation, 2016)(Kalman, 2014).

Unhappily there is no software component made perfect and there is actually no way to know in advance all the vulnerabilities of a software component. It should be enough in most cases to avoid those components which are really plagued with serious vulnerabilities (e.g. Apache CXF Authentication Bypass vulnerability in the Apache CXF framework or the Spring Remote Code Execution vulnerability from the Spring Expression Language component (The OWASP Foundation, 2016)).

The good part of this vulnerability is the fact that many components used by the NoSQL based applications are in their early stages, so they and their vulnerabilities are not known enough to be exploited.

The bad part is that, well, many components used by the NoSQL based applications are in their early stages, so they and their vulnerabilities are not known enough to be patched or avoided, and this not known for the moment vulnerabilities will surface at a later time when, maybe, plenty of applications will be based on these components.

DOI: 10.24818/18423264/53.3.19.07

J. Unvalidated Redirects and Forwards

Web applications may redirect or forward users to other pages and websites, and sometimes these redirects are based on untrusted / unverified data. This is one of the well-known attack vectors which can be used to send victims to phishing or malware sites, or to access unauthorized pages(The OWASP Foundation, 2016)(Kalman, 2014).

At least, for this type of vulnerability, there is nothing specific to NoSQL based applications. One should expect the same level of risk on this type of vulnerability for both relational and NoSQL based applications. The difference would be the fact that the relational DB based applications are still the vast majority (we will discuss this factor in detail a little bit later in the Trends section) so we should expect that, at least for the near future, attacks based on this vulnerability will be in greater number targeted to relational DB based applications.

3. New classes of security vulnerabilities

A. Schema based attacks

Many of the NoSQL solutions have a new approach on creating database structures – new schemas will be created at the moment they are needed – inserting data in a schema that does not exists will automatically create the schema. On top of that, most of the times the new schema is not access-protected in any way(Chow, 2013).

While not offering access to already existing data from other schemas, this vulnerability can be used for various types of attacks, such as a DDoS attack designed to fill up the existing storage by creating new schemas and loading them with garbage data.

B. Product specific issues

Each NoSQL solution, being based on a particular technology, will have its own security issues, as expected. we will simply exemplify this with some of the vulnerabilities of a few products.

MongoDB specific security issues(Chow, 2013):

- The run() command can act as shell;
- Significant information can be extracted directly from the startup_log from the local collection (pid, OS details, paths);
- An unsupervised sniff tool (mongosniff) is included in default MongoDb installation. This tool can be used for tracing / sniffing the database activity in real time.

DOI: 10.24818/18423264/53.3.19.07

- Unauthorized access to a MongoDB instance or cluster when using LDAP authentication (corrected in version 3.0.7)
- Remotely trigger a denial of service (crash) due to failure to check for missing value (corrected in version 3.0.1)
- Remotely trigger a denial of service (crash) via a specially crafted regular expression (corrected in version 2.6.9 and 3.0.1)
- A specially crafted, malformed BSON message may trigger an uncaught exception in the server, resulting in a loss of availability (corrected in version 2.6.8 and 2.4.13)
- Remotely trigger a crash when X.509 authentication is enabled (corrected in version 2.6.2)
- Information disclosure of user credentials (corrected in version 2.6.1)
- Improperly grant user system privileges on databases other than local (corrected in version 2.4.5, 2.5.1)
- Remotely triggered segmentation fault in JavaScript engine (corrected in version 2.4.5, 2.5.1)
- It is possible to create documents that collide with JavaScript functions when fetched using the mongo shell

CouchDBspecific security issues (Chow, 2013)(Apache CouchDB, 2016):

- The HTTP / REST API is exposed by default
- Apache CouchDB Timing Attack Vulnerability
- Information disclosure via unescaped backslashes in URLs on Windows
- JSONP arbitrary code execution with Adobe Flash
- DOM based Cross-Site Scripting via Futon UI
- DoS (CPU and memory consumption) via the count parameter to /_uuids

The above examples shouldn't be too scary as similar lists are (or should be) existing for every DB solution ever made (be it relational or NoSQL). The only lesson to be learned from here is that the NoSQL solutions are not exempted from such vulnerabilities only because they are newer.

A brief description of the above listed vulnerabilities, compared to their homologues in the relational database world, can be found in the following table:

Vulnerability	Relational database applicable?	NoSQL database applicable?
SQL injection	Yes. A classic vulnerability, with multiple counteract / mitigation measures available.	No.
OS and LDAP injection	Yes. A classic vulnerability, with multiple mitigation measures available.	Yes, it's not DBS dependent. It's related to the authentication / authorization method, so

DOI: 10.24818/18423264/53.3.19.07

		the same mitigation measures as for the relational DBS are valid.
JSON injection	No. Relational database system are not usually dependent on JSON data transmissions. If, by exception, such an instance will occur in some app, the mentioned app will be vulnerable.	Yes. Same counteract / mitigation measures as for SQL injection should be effective in some of the occurrences, but not in all
Broken authentication and session management	For some products. Most notable relational database offer strong authentication and session management, but the same is not always the case for smaller-foot DBS such as MS Access, SQL Compact and so on.	For many products. Many NoSQL DBS are still in infant or early stages.
Cross-Site Scripting (XSS)	For some applications based on relational DBS. Especially when the app includes some JavaScript interface.	For some applications based on NoSQL DBS. Especially when the app includes some JavaScript interface.
Insecure direct object references	Yes. A classic vulnerability, with multiple counteract / mitigation measures available.	Yes. Counteract / mitigation measures available, but highly dependent on the DB administrator's skills.
Missing function level access control	Yes. A classic vulnerability, with multiple counteract / mitigation measures available.	Yes. Counteract / mitigation measures available, but highly dependent on the DB developer's skills.
Cross-Site Request Forgery (CSRF)	Yes. Especially when HTTP / REST API's are provided for the respective DBS.	Yes. Especially when HTTP / REST API's are provided for the respective DBS.
Using known vulnerable components	Yes. Less prevalent for mature DBS products.	Yes. More prevalent as many NoSQL DBS are still note mature enough.
Unvalidated redirects and forwards	Yes. More occurrences foreseeable as the relational DBS are dominating the market.	Yes. Less occurrences foreseeable as the relational DBS are dominating the market.

DOI: 10.24818/18423264/53.3.19.07

Schema based attacks	No.	Yes.
Product specific issues	Yes. Less prevalent for mature DBS products.	Yes. More prevalent as many NoSQL DBS are still note mature enough.

4. Counteracts and mititgation

A vulnerability review such is the one from this paper will not be complete without a few words about how to counteract / mitigate the many listed issues.

A good starting point should be again The OWASP Top 10 Application Security Risks – 2013 list(The OWASP Foundation, 2016). This list is not only providing classes of known vulnerabilities but also a large amount of prevention measures, advices and best practices.

Note. The same organization is providing supplementary info via so called cheat sheets - SQL Injection Prevention Cheat Sheet(The OWASP Foundation, 2016), XSS (Cross Site Scripting) Prevention Cheat Sheet(The OWASP Foundation, 2016) and Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet(The OWASP Foundation, 2016), to name a few ones[1].

A few proposed counteracts:

- For injection vulnerabilities: native encoding, static code analysis, Dynamic Application Security Testing (DAST), Interactive Application Security Testing (IAST)(Ron, et al., 2015), use of safe API which avoids the use of the interpreter entirely or provides a parameterized interface, careful inspection of escaped special characters using the specific escape syntax for that interpreter(The OWASP Foundation, 2016).
- For REST API exposure and CSRF attacks: control the requests, limiting their format, make sure JSONP and CORS are disabled in the server API to make sure that no actions can be made directly from a browser(Ron, et al., 2015).
- For Access Control and Prevention of Privilege Escalation: proper authentication and RBAC authorization, proper privilege isolation(Ron, et al., 2015)
- For Sensitive Data Exposure: make sure you encrypt all sensitive data at rest and in transit, don't store sensitive data unnecessarily / discard it as soon as possible, ensure strong standard algorithms and strong keys are used, and proper key management is in place, ensure passwords are stored with an algorithm specifically designed for password protection, such as bcrypt, PBKDF2, or scrypt, Disable autocomplete on forms collecting

[1] The complete list can be found at https://www.owasp.org/index.php/Cheat_Sheets

DOI: 10.24818/18423264/53.3.19.07

sensitive data and disable caching for pages that contain sensitive data(The OWASP Foundation, 2016).

5. Trends

A decent vulnerability review should not be over without a forecast of future trends.

At this moment (May 2016) the relational DBMS are covering 81.6% of the market (popularity wise), the rest of 18.4% being divided between various flavors of NoSQL solutions (2.9% for the wide column stores, 3.4% for the key-value stores, 6.8% for the document stores).

In the last three years, various types of NoSQL solutions have seen raises of 19.6-420.25% in popularity (179.13-241.65% for the three main subtypes – key-values stores, document stores and wide column stores).In the last two years, the same NoSQL solutions raises in popularity were of 0.36-69.49% (40.75-53.13% for the three main subtypes).The last year have seen even lower grows, NoSQL popularity changes being in the range of -19.71-20.46% (10.68-11.93% for the three main subtypes).At the same time (the last three / two / one years) the relational solutions popularity remained mostly constant(Anon., 2016).

Looking at these numbers, we can safely assume that initial hype period is almost over and the trend for the following few years will be one of slow(er) growth for the NoSQL solutions, with a stabilization at about 20-22% percent popularity share (a nonlinear regression analysis model chosen for best fitness, applied on the above mentioned values, gave a 5.5-7.3% growth for the three main subtypes in 2016, a 2.6-3.3% growth for the three main subtypes in 2017 and finally a 0.3-1% growth in 2018; of course, such a forecast is only a game with numbers for there are a lot of factors which can influence the market evolution).

Starting from the results of this game with numbers and from the rule of thumb that the number of discovered vulnerabilities is related with the product popularity, one should not see a great increase in the number of vulnerabilities related to the NoSQL solutions for the following few years.

6. Statistics

It makes sense to compare the NoSQL databases security to the relational databases security from a statistical point of view (e.g. number of known vulnerabilities, number of applied vulnerability patches etc.), but only if we also take into consideration two other factors, the difference in market share between the two categories (a lower market share means both a lower number of installations and a lower effort for detecting new vulnerabilities by the all interested parties, be them “white” or “black”), and the difference in “age” (an “older” product had enough time to accumulate a larger number of discovered vulnerabilities).

DOI: 10.24818/18423264/53.3.19.07

The following table takes into account the data available in Common Vulnerabilities and Exposures List[2] for five relational and five NoSQL database products. While the sheer number of discovered vulnerabilities is in no way an indicator of a low level of security, we can accept as a base for a possible analysis the quotient between the number of discovered vulnerabilities and the level of interest received by the products (as measured, by example by the DB-engines ranking[3]):

DBS	2019 vulnerabilities #	2018 vulnerabilities #	2017 vulnerabilities #	2016 vulnerabilities #	2015 vulnerabilities #	2014 vulnerabilities #	Total vulnerabilities #	DB-engines score	2018-2019 vulnerabilities # /score
Oracle Database	9	12	11	30	32	46	140	1288	0.0162
MySQL	67	134	118	124	90	79	612	1218	0.1648
MS SQL Server	1	5	1	12	8	2	29	1072	0.0055
PostgreSQL	1	13	17	14	10	16	71	479	0.0292
MongoDB	0	5	3	3	5	2	18	408	0.0122
IBM DB2	4	32	13	5	5	12	71	174	0.2063
Elasticsearch	1	8	3	2	8	3	25	148	0.0605
Redis	3	8	3	6	4	1	25	148	0.0741
Cassandra	0	2	0	0	2	0	4	126	0.0159
Hbase	1	1	0	1	1	0	4	60	0.0334
Total relational	**82**	**196**	**160**	**185**	**145**	**155**	**923**	**4233**	**0.0656**
Total NoSQL	**5**	**24**	**9**	**12**	**20**	**6**	**76**	**891**	**0.0325**

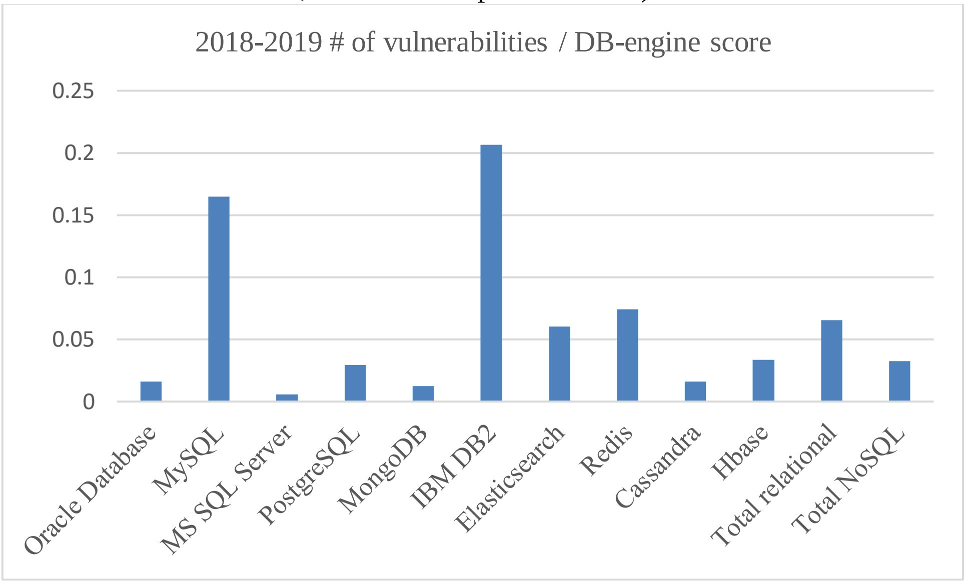
If the quotient between the number of discovered vulnerabilities and the level of interest given to the products can be taken as some sort of indicator of the level of security, as proposed above, several possible (contradictory!) conclusions can be drawn,(and maybe analyzed in further studies) such as:

- The NoSQL database products benefit from the fact that they are "young" anddo not follow upward compatibility so their code is less bloated and more secure.
- There are too few NoSQL related discovered vulnerabilities, so there must be others which are not discovered yet, which indicates towards a lower level of security.

[2]http://cve.mitre.org
[3]https://db-engines.com/en/ranking

DOI: 10.24818/18423264/53.3.19.07

Not taking into account any possible interpretations, we found large variations in the ratio proposed above (and even larger variations in the number of discovered vulnerabilities, as seen in the previous table):



An important observation to be taken into account: while researching for the present paper, the authors found no relevant study or statistics either informing the managers on the issues related to NoSQL databases security, or describing their perception of the subject. It seems that NoSQL databases security, as an important element of information systems security management, is not at this moment a well-documented subject and may require time and efforts in order to become a familiar concept for managers.

7. Conclusions

Most of the NoSQL DBMS are in early production stages and as such, are prone to a lot of improvement. This is especially true on the security side, as the initial approaches of the majority of NoSQL solution vendors where directed mostly toward performance, the security issues being put aside for some time.

Recent developments are about to change this state as the preeminent NoSQL solutions are already closing a full maturity stage. Such developments are also expected to occur in the following years regarding the human resources involved - mainly administrators and developers, but other categories of staff should also be trained appropriately about the characteristics of NoSQL solutions.

Even not taking into account this maturity issue, as any other complex software product, NoSQL solutions are and will be having various security vulnerabilities and these must be known, taken into account, counteracted and

DOI: 10.24818/18423264/53.3.19.07

mitigated as much as possible in order to achieve a relative state of security for the organization informational assets.

As the current paper is far from covering the considered subject, the authors feel that a few furthering reading suggestions would be welcome. As such, the reader may also like to look at the following papers:

- Sethuraman Srinivas, Archana Nair, "Security maturity in NoSQL databases - are they secure enough to haul the modern IT applications?"(Srinivas & Nair, 2015)
- Lior Okman, Nurit Gal-Oz, Yaron Gonen, Ehud Gudes , Jenny Abramov, "Security Issues in NoSQL Databases"(Okman, et al., 2011)
- Anam Zahid, Rahat Masood, Muhammad Awais Shibli, "Security of sharded NoSQL databases: A comparative analysis"(Zahid, et al., 2014)
- Iván Arce, et al., "Avoiding the Top 10 Software Security Design Flaws"(Arce, et al., 2014).

REFERENCES

[1] **Anon (2016),** ***DBMS Popularity Broken down by Database Model***. [Online] Available at: http://db-engines.com/en/ranking_categories [Accessed 28 April 2018];

[2] **Apache CouchDB (2016),** ***Apache CouchDB 2.0.0 Documentation - 16.*** *Security Issues Information.* [Online] Available at: http://docs.couchdb.org/en/1.6.1/cve/ [Accessed 29 April 2018].;

[3] **Arce, I. et al. (2014),** ***Avoiding the Top 10 Software Security Design Flaws,*** **s.l**.: IEEE Center for Secure Design;

[4] **Butturini, R. (2014),** ***Making Mongo Cry: NoSQL for Penetration Testers - Defcon 22 Wall of Sheep Presentation Slides***. [Online] Available at: http://www.nosqlmap.net/index.html [Accessed 28 April 2018];

[5] **Chow, M. (2013),** ***Abusing NoSQL Databases.*** Las Vegas, s.n.

[6] **DB Engines, (2016),** ***DB-Engines Ranking***. [Online] Available at: http://db-engines.com/en/ranking [Accessed 28 April 2018];

[7] **Edlich, S. (2012),** ***NoSQL Database.*** [Online] Available at: http://nosql-database.org/ [Accessed 28 April 2018];

[8] **Kadebu, P. & Mapanga, I. (2014),** ***A Security Requirements Perspective towards a Secured NoSQL Database Environment***. Delphi, s.n.;

DOI: 10.24818/18423264/53.3.19.07

[9] **Kalman, G. (2014),** ***10 Most Common Web Security Vulnerabilities.*** [Online]
Available at: https://www.toptal.com/security/10-most-common-web-security-vulnerabilities
[Accessed 29 April 2018];

[10] **Kirkpatrick, D. (2013),** ***Mongodb - Security Weaknesses in a typical NoSQL Database***. [Online]
Available at: https://www.trustwave.com/Resources/SpiderLabs-Blog/Mongodb---Security-Weaknesses-in-a-typical-NoSQL-database/
[Accessed 29 April 2018];

[11] **Martin, B., Brown, M., Paller, A. & Kirby, D.(2011),** ***CWE/SANS Top 25 Most Dangerous Software Errors***. [Online]
Available at: http://cwe.mitre.org/top25/
[Accessed 28 April 2018];

[12] **Martin, B., Brown, M., Paller, A. & Kirby, D. (2011),** ***On the Cusp: Other Weaknesses to Consider.*** [Online]
Available at: http://cwe.mitre.org/top25/archive/2011/2011_onthecusp.html
[Accessed 28 April 2018].

[13] **Okman, L. et al. (2011),** ***Security Issues in NoSQL Databases***. Changsha, s.n.;

[14] **Oku, K. (2014),** ***The JSON SQL Injection Vulnerability***. [Online]
Available at: http://blog.kazuhooku.com/2014/07/the-json-sql-injection-vulnerability.html
[Accessed 28 April 2018];

[15] **Ron, A., Shulman-Peleg, A. & Bronshtein, E. (2015),** ***No SQL, No Injection?.*** San Jose, s.n.;

[16] **Rossi, B.(2015),** ***Major security alert as 40,000 MongoDB databases left unsecured on the internet***. [Online]
Available at: http://www.information-age.com/technology/security/123459001/major-security-alert-40000-mongodb-databases-left-unsecured-internet
[Accessed 28 April 2018];

[17] **Singh, A. (2016),** ***5 Business Challenges That May Backfire Your NoSQL Strategy***. [Online]
Available at: http://www.algoworks.com/blog/business-challenges-that-backfire-nosql-strategy/
[Accessed 29 April 2018];

[18] **Srinivas, S. & Nair, A. (2015),** ***Security Maturity in NoSQL Databases - Are they Secure enough to Haul the Modern IT Applications?.*** Kochi, s.n.;

[19] **Sullivan, B. (2011),** ***Server-side JavaScript Injection - Attacking and Defending NoSQL and Node****. JS.* Las Vegas, s.n.

DOI: 10.24818/18423264/53.3.19.07

[20] **The OWASP Foundation (2016<** ***Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet.*** **[**Online]
Available at: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_%28CSRF%29_Prevention_Cheat_Sheet
[Accessed 29 April 2018];

[21] T**he OWASP Foundation (2016),** ***OWASP Top Ten Project***. [Online]
Available at: owasptop10.googlecode.com/files/OWASP%20Top%2010%20-%202013.pdf
[Accessed 28 April 2018];

[22] **The OWASP Foundation (2016),** ***SQL Injection Prevention Cheat Sheet.***
[Online]
Available at:
https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
[Accessed 29 April 2018];

[23] **The OWASP Foundation (2016),** ***XSS (Cross Site Scripting) Prevention Cheat Sheet***. [Online]
Available at:
https://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Cheat_Sheet
[Accessed 29 April 2018];

[24] Y**egulalp, S. (2015),** ***3 security pain points NoSQL must remedy.*** **[**Online]
Available at: http://www.infoworld.com/article/2884320/security/3-security-pain-points-nosql.html
[Accessed 29 April 2018];

[25] **Zahid, A., Masood, R. & Shibli, M. A. (2014),** ***Security of sharded NoSQL databases: A comparative analysis***. Rawalpindi, s.n.

DOI: 10.24818/18423264/53.3.19.07