(In) Security in Graph Databases
Analysis and Data Leaks

Miguel Hernández Boza[1] and Alfonso Muñoz Muñoz[2]
[1]*Innovation 4 Security - BBVA , Av. Burgos 16D 28036, Madrid, Spain*
[2]*Innovation 4 Security Labs - BBVA, Av. Burgos 16D 28036, Madrid, Spain*

Keywords: Graph Database, Privacy, Neo4j, OrientDB, Grafscan.

Abstract: Graph databases are an *emerging* technology useful in the field of cybersecurity, especially in the detection of new threats based on the correlation of diverse sources of information. In our research, we had reviewed the design of the most widespread graph databases, for example Neo4J and OrientDB, detecting several security problems, improper default configurations and leaks, scanning the Internet during 9 months. To repeat our proofs, we are releasing the first hacking tool for testing and detecting (in) secure graph databases, GraFScaN.

1 INTRODUCTION

The current *knowledge society* has a clear dependence on the creation of new contents, in multiple formats, that will be consumed anywhere and at any time. This need as a society generates that the creation of new data increases every day. This can be observed in any social network, blog or instant messaging application at present: new photos, new videos, new links, etc. In this environment, the form and the means used for its storage and subsequent processing are critical, especially in today's society where free and massive services are required, where the presence of advertisements and the extraction of information through the analytics techniques or data mining are crucial for new business models.

In this context, the management of the temporal component in the creation, storage or consumption of such information, especially with the addition of new sources of information, is not negligible. This problem is significant in fields such as astronomy or genomics, and of course in information search technologies on the Internet, for example search engines or financial systems (Zachary, 2015).

For all these reasons, Big Data technologies have been focusing on the collection, storage, search, analysis and visualization of large amounts of data. Such is the interest of these technologies that in the last decade a multitude of tools have been developed for the treatment of large volumes of information centered on three types of data: **structured data** (data well defined length and format, typically stored in relational databases), **unstructured data** (data without specific format and is stored as it was collected) and **semi-structured data** (data that is not limited to certain fields but has markers to separate information, for example the JSON standard).

At the same time, the different types of data can be stored considering different formats. The most typical are: **key-value storage** (they are stored in a similar way to a data dictionary where the data is accessed from a single key), **document storage** (the structure of the data stored are very similar to key-value storage, differing in the data they keep), **column-oriented storage** (it is oriented to scale horizontally with what allows to save different objects and attributes on a same key) and **storage in graph**. The latter breaks with the idea of tables and are based on graph theory, where it states that information are nodes and relations between information are edges. Its greater use is contemplated in cases of relating large amounts of highly variable information, for example in social networks.

In this point, and after a brief understanding of the different types of data and their nature, it was opportune to delve into a specific type of data storage, the graph databases. Its ability to relate information defines it as an excellent technology in different areas of cybersecurity (fraud detection, authentication process management, threat management, etc.) and, therefore, it is necessary to understand its design and analyse if they present security failures that could affect other systems within an organization.

Boza, M. and Muñoz, A.
(In) Security in Graph Databases - Analysis and Data Leaks.
DOI: 10.5220/0006419003030310
In *Proceedings of the 14th International Joint Conference on e-Business and Telecommunications (ICETE 2017) - Volume 4: SECRYPT*, pages 303-310
ISBN: 978-989-758-259-2

2 GRAPH DATABASES. ANALYSIS AND VISUALIZATION

2.1 Introduction

The beginning of graph theory (Godsil, 2001) took place in 1736, in an article by Leonhard Euler (Euler, 1736). The work arose from a problem known as the Königsberg bridge problem. The problem was to go through all the bridges once and go back to the same starting point.

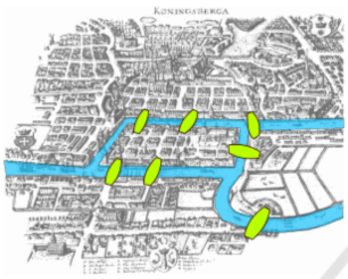
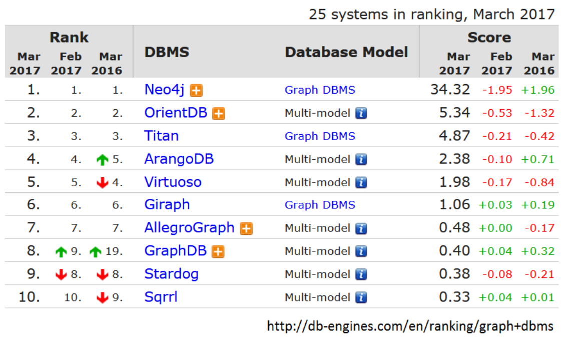


Figure 1: The bridge problem of Königsberg.

Euler proved that it was not possible. To do this, it replaced each starting zone by a point and each bridge by an arc, thus creating the first graph, designed to solve a problem. This case gave rise to graph theory.

Nowadays, the current technology has allowed to take these principles to new type of database, in this case oriented to graphs. The information is stored as nodes or entities and the relationships between them allow to obtain an additional information of great value. Although, it is true that this technology is relatively recent, the most significant technological milestones start from 2013, however there are already certain databases that can be used in real systems with success. The most famous graph database, by its evolution and community, is Neo4j, followed by far by OrientDB (Figure 2).



25 systems in ranking, March 2017

Rank			DBMS	Database Model	Score		
Mar 2017	Feb 2017	Mar 2016			Mar 2017	Feb 2017	Mar 2016
1.	1.	1.	Neo4j	Graph DBMS	34.32	-1.95	+1.96
2.	2.	2.	OrientDB	Multi-model	5.34	-0.53	-1.32
3.	3.	3.	Titan	Graph DBMS	4.87	-0.21	-0.42
4.	4.	↑ 5.	ArangoDB	Multi-model	2.38	-0.10	+0.71
5.	5.	↓ 4.	Virtuoso	Multi-model	1.98	-0.17	-0.84
6.	6.	6.	Giraph	Graph DBMS	1.06	+0.03	+0.19
7.	7.	7.	AllegroGraph	Multi-model	0.48	+0.00	-0.17
8.	↑ 9.	↑ 19.	GraphDB	Multi-model	0.40	+0.04	+0.32
9.	↓ 8.	↓ 8.	Stardog	Multi-model	0.38	-0.08	-0.21
10.	10.	↓ 9.	Sqrrl	Multi-model	0.33	+0.04	+0.01

http://db-engines.com/en/ranking/graph+dbms

Figure 2: Ranking of graph DBMS (Db-engines, 2017).

Up to our knowledge, scientific literature did not report yet any research about the specific topics covered in this paper. For that reason, in the following sections, the security of the most important graph databases will be deepened.

2.2 Case Studies: Neo4j and OrientDB

Neo4j (Todd, 2009) is a graph-oriented database and free software, implemented in Java. This database stores the information natively in nodes and relations instead of tables. Among its features, it has the possibility of adding several tags to the nodes and an unique one to the relations, besides being able to add properties or indexes. The biggest feature of this database is its query language, *Cypher*.

Cypher (Cypher, 2017) is a declarative language to describe patterns in graphs visually using *ASCII-Art* syntax. It allows to indicate what one wants to consult, to insert, to update or to eliminate of the data in the graph without the necessity to describe it with precision. To select the nodes the parentheses are used because of their similarity to the circles, for example, a node p would be represented as *(p)*. In addition, you can add a series of tags that facilitate the search of the nodes, as it can be *(p: Person)*. Within each node, a property can be accessed by using the dot, for example *p.name*. All of these values are case sensitive. Therefore, the general structure of a query can be seen as follows:

```
MATCH (node:Label) RETURN node.property
MATCH (node1:Label1)-->(node2:Label2)
WHERE node1.propertyA = {value}
RETURN node2.propertyA, node2.propertyB
```

You can consult relations between nodes using arrows, which can be directional, and additional information can be added using brackets inside the arrow. An example of the structure:

```
MATCH (n1:Label1)-[rel:TYPE]->(n2:Label2)
WHERE rel.property > {value}
RETURN rel.property, type(rel)
```

It is advisable to consult the technical manual of this database to understand in detail this query language (Cypher, 2017).

OrientDB (Orientdb, 2017) is a non-relational open source database management system written in Java. This system supports documents, key / value, and object models but relationships are managed as in graph-oriented databases with direct connections between registers.

OrientDB incorporates a key-value and object-oriented search engine. Its query language is SQL, with some extra functionality to enable functionality with graphs. Queries are case sensitive. The SQL engine automatically recognizes whether any of the indexes can be used to improve execution, although can be explicitly added.

3 SECURITY IN Neo4j. STRENGTHS AND LIMITATIONS

3.1 Versions

In 2013 (Neo4j, 2013) Neo4j presents the first version 1.9 with serious design flaws, including the possibility of executing remote code (CVE-Neo4j, 2013), using a public exploit, CVE-2013-7259. At the end of that year appeared the first version 2.0 (Neo4j 2.0, 2013) where the visual section began to have more importance. It was solved the problem of remote code execution and was added the possibility of having HTTPS. Until 2016 it was not evolved to new versions.

In April 2016, in Graph Connect, version 3.0 was introduced (Neo4j 3.0, 2016), adding a multitude of functionalities, new connection protocols and the web administration panel disappeared. This last point introduces the possibility to an attacker, by using fingerprinting techniques, know the version of a Neo4j database without having to be authenticated in it. If the panel exists the version of the database will be less than 3.0 and if not higher. At the end of 2016 appeared the new version 3.1 with the great novelty of the implementation of user management.

The configuration and design of each of these versions can be used by an attacker to try to access a database without credentials in it and to be able to insert, steal, manipulate or delete information as will be shown below.

3.2 Configuration

The default configuration and version management of this graph database is vital from an attacker's point of view. Therefore, the configuration (configuration path and files) is analysed depending on the versions:

Version 2.x: In this case the configuration information is located in the following path: {*path-neo4j*}*/config/neo4j-server.properties*, within which the configuration settings to be detailed.

#org.neo4j.server.webserver.address=0.0.0.0

This line configures which addresses can access the service running Neo4j. The reader can observe that by default is commented out and only the local user can connect to it. On the contrary, removing the comment would mean that if you expose your IP and is accessible from the Internet, that the service (database) would be exposed and any user (legitimate or otherwise) would have visibility of the Neo4j database.

Also, it is possible to disable the login password authentication in user's access (default enabled). It is not recommended in any case but in some connectors to the database is recommended because there is no implementation of the authentication process (Neo4j connector, 2017).

dbms.security.auth_enabled=true

The default port for remote connection is port 7474/7473. There is the possibility of changing this port for less exposure to unsophisticated attackers. Obviously, it is not a sophisticated protection measure.

org.neo4j.server.webserver.port=7474
org.neo4j.server.webserver.https.port=7473

In some versions for Windows operating system is only possible to connect via HTTP, appearing the HTTPS line commented (Neo4j conf, 2017).

Neo4j implements a remote shell on the default port 1337. This configuration is for the *remote* shell to the database, to be able to query it by *Cypher*. This shell presents security problems because an user can connect without authentication. Luckily, it is not enabled by default and it is only possible to access through the computer itself. It is important to monitor the change of this configuration.

#remote_shell_enabled=true
#remote_shell_host=127.0.0.1
#remote_shell_port=1337

Version 3.x: Unlike the previous versions, the configuration of these comes enclosed within a same file, with the same parameters as the previous version. The configuration is in the following path: {*path-neo4j*}*/conf/neo4j.conf*.

3.3 Protection and Storage of the Password

In all versions, the key storage is stored in a file on the following path: *$path-neo4j$/data/dbms/auth*. This file is created when Neo4j is started the first time and at the end of the file appears: *password_change_required*. When the user by the browser access to the portal, it asks for the initial authentication (default-user *neo4j / neo4j*) and asks to change it. The limitation of the password is that it contains at least one character, but it does not have a maximum character and complexity limit, also it is not allow to put the same default password *neo4j*. It is certainly not the best robust password management policy.

The password is stored using the *sha256* + *salt* algorithm, that without being a bad option it is not the best mechanism against dictionary attacks / rainbow-tables / brute force off-line attacks. It would be advisable to use PBKDF2 (Burt, 2015).

3.4 User Authentication. Design Failures

The system that implements Neo4j at the time of authenticating is an HTTP Basic (Basic Auth, 2017). The key of a user is sent to the database, encoded in base64, in the header of each request that requires authentication. This is especially sensitive when setting up the connection via HTTP.

The authentication system has protection against brute-force attacks that blocks by IP source. Neo4j detects the third attempt to access with bad credentials by entering a constant delay of 5 seconds to accept the following request (key stretching). This protection is simple to override by using different intermediate devices or *proxies* to change the source IP and avoid this restriction. In the tests performed, counting the network delay, using 5 IPs (5 open proxies) it is possible to run a dictionary / brute force attack without limitations.

Additionally, another attack from a design flaw was detected. Neo4J versions 2.x and 3.x allow access to an *end-point* that allows changing the user's key if the current key is provided. This is a problem since an attacker could send requests by trying different current keys and force the change always to the same key (*new key*). By designing the database, an attacker does not need to receive a response from this request, which means that an intermediate system (e.g. IoT) could be used without having to control the response. The attacker from time to time, does not know when a request will hit the key change, will connect to the database, for example via Tor, with the *new key*. If the brute force attack was successful you will have full access to the database.

3.5 Licenses

Neo4j has two types of licenses (Neo4j lic, 2017), *Community* (free) and *Enterprise* (paid). The *Enterprise* license includes a twenty-four-hour support system, high availability, backups around the maintenance and a whole series of security measures (logs, limitation of execution time, etc.). The difference between licenses is a security problem added to Neo4j. The cost is a problem in the use of the *Enterprise* license and the *Community* license does not present basic security options such as a *log*. This will facilitate attacks and leaks of information.

3.6 Injections

If access to Neo4j is achieved, either by not having authentication enabled, incorrect validation in the input of information to the database (would facilitate injection of queries *cypher*) or by discovery of the credentials, we would have full access to the information of the database. At this point, it makes sense to analyse how the use of *Cypher* language can allow different injections with security implications. The most classic would be those that can be done in the most common languages, such as *SQL*, download, delete or edit the information contained in the database. In addition, it is proposed to analyse the impact of different queries that would allow denial of service or new forms of encryption of the entire database as a new *ransomware* (Mehmood, 2016).

Disc Denial

One of the techniques investigated to deny the use of storage (disk) would be, for example, by using a request that begins to import data to occupy all available free space, data that can be imported from external sources, via url. This attack takes advantage of the current configuration in which the amount of disk that the database can reserve is not limited. The example query to import an external file would be the following:

USING PERIODIC COMMIT 1000 LOAD CSV FROM URL AS row CREATE (A:N1{a:row[0]})-[:RE]->(B:N2{b:row[1]});

It is possible to accelerate this type of attacks and get the disk to fill before, using a misuse of Neo4j (Cypher T, 2017) creating indexes in the database without control, because the space needed for each index is greater for each data that you have to index. Therefore, if you create a large volume of indexes, it is possible to do it in a single query, you can make the data import operation more efficient and fill the disk before.

RAM Denial

The attacks investigated to deny the use of RAM, causing a maximum reservation of the same, take advantage of the fact that when loading information from *Cypher* accumulates all the entities and relationships that are created to insert them all to the same time. A possible attack can force, by one or more requests, to store a large amount of information in memory by importing a huge number of nodes through a loop, using the term that provides *Cypher FOR EACH* for them. An example of *request* would be:

FOREACH (x in range(1,10000000000000) — CREATE (:Person {name:name+x, age: x%100}));

Neo4j allows to limit the execution time of queries of this type in the database. Although this feature could be used as a countermeasure to this type of attacks, it would be necessary to know very well the queries and data in the database (data that can be dynamic) so that this alleged countermeasure does not impede the normal operation with the database.

Ransomware

Currently, ransomware attacks on databases are based on the possibility of exploiting a vulnerability or configuration failure to export-import the database and encrypt its contents to request a later rescue (retrieve the information in clear). A significant example of this has been the recent breach of MongoDB databases (Pauli, D, 2017).

In addition to this possibility, which is viable if one has access to import and export the Neo4j database (database without authentication, query injection or authentication violated), the possibility of new attacks of *ransomware* was analysed, taking advantage of specific features of this database. In fact, it was observed how the *Cypher* language itself and its query engine allows not only replace values (with the function *replace*), which would allow, for example, a homophonic cipher (Hammer, 1981), if not, in addition, eliminate the relations between the nodes, with which in practice the database would be useless (although it is reversible on the part of the attacker). For example, the following query in *Cypher* encrypts the *prop1* of the graph by doing concatenation of replacements:

MATCH (n) where n.prop1 set n.prop1 = replace(replace(a,81)...) z,12)

To remove the relationships, it is necessary to keep a copy of these to be able to reverse the process. This can be done with the following query:

MATCH (n)-[r]->(m) RETURN ID(n), ID(m), type(r), properties(r)

And, finally, the elimination of the relationships that exist in the database:

MATCH ()-[r]->() DELETE r

It is noteworthy that most databases exposed on the Internet (as will be seen later) were licensed Community (without the option of logs) so that Neo4j itself (without the help of external technology) could not know the queries made by the attacker or protect itself easily even if the cryptographic attack is not very sophisticated. In theory, from the calculations made, when doing the research, it would have been possible to encrypt all databases exposed on the Internet without security (more than 400) in less than 10 minutes.

At the following link we published a demo video of how an attack of this type would work: *https://youtu.be/jwXS3muRkx0.*

4 SECURITY IN OrientDB. STRENGTHS AND LIMITATIONS

Unlike Neo4j, OrientDB is a server that facilitates the management of different graph-oriented databases.

4.1 Versions

There are a lot of versions, which are appearing monthly. The Table 1 describes the number of versions per year (new features are constantly added). Versions prior to 2.0.15 and 2.1.1 have the following CVE: *CVE-2015-2918* (remote attack of click-hacking), *CVE-2015-2913* (to predict session numbers) and *CVE-2015-2912* (CSRF attacks).

Table 1: Versions of OrientDB.

		Years				
		2013	2014	2015	2016	2017
Versions	1.x	9	4	0	0	0
	2.0	0	2	7	1	0
	2.1	0	0	4	18	0
	2.2	0	0	0	14	2
	Total	9	6	11	33	2

4.2 Configuration

The configuration information is stored in the installation folder, {*path-OrientDB*}*/config*, where the following files are located:

orientdb-server-config.xml. In this file are the necessary parameters to establish connection and the users of the server of OrientDB with the summaries of their passwords. This configuration is by default, and unlike Neo4j, allowing the connection from any source IP to the OrientDB server, exposing it to the Internet. The strange thing is that the documentation indicates the opposite (OrientDB expo, 2017). It is significant, in addition, how to work with the user *root*, making it more vulnerable against brute-force attacks, especially since it is possible to remotely connect to the root account.

security.json. This file shows the security settings of the OrientDB server, changes can be made to improve security. Highlight that three default users are created for access to any database. The users created by default are: *admin:admin, writer:writer and reader:reader*. Similarly, a default database, named *Gratefuldeadconcerts*, is created.

Finally, authentication is enabled by default just like in Neo4j and it would be necessary to disable the credentials manually.

4.3 User Authentication. Design Failures

Initially, when the server is started, it asks to be assigned a password for the user *root*, which will have full access to all the databases and the management portal. If the password is left blank, a random is created.

The authentication is based on *HTTP Basic* as in Neo4j, in this case you have credentials for each database that is inside the server. As seen in the configuration, if not changed, three users are generated to access each database, which initially have a default credentials to try to access. It is possible to manage from the browser the creation of more users and assign them the roles that appear in the documentation (OrientDB rol, 2017), some of them allow to crosswise access to all the databases of the server.

The OrientDB server does not deploy any system against brute-force attacks, either to attack access to a particular database or to take root credentials and access them all. This point is significant especially if it joins an additional vulnerability detected. The documentation explains that from the server of OrientDB can not list the databases that are inside the server without authentication (AuthOrientDB, 2017), but if a request is sent to the *end-point*/listdatabases, for example *http://10.0.0.1:2480/listdatabases*, you can see that it returns a list with the names of the databases, and in the response header, the version of the OrientDB server is sent, all without requiring credentials. All this greatly simplifies the *fingerprinting*.

4.4 Licenses

As in the case of Neo4j, OrientDB has two types of licenses, *Community* and *Enterprise*. This last one adds different functionalities for the management of the databases but the security characteristics are equal in both licenses, which is a better defensive approach than in Neo4j, at least if it thinks in global terms.

5 EXHIBITION OF PRIVATE INFORMATION. DATA LEAKS AND INTERNET

In order to verify the impact of some of the security problems detected, among the most spread graph databases, a verification process was required. With this purpose, a specific fingerprinting tool, called GraFScaN (Hernández, M; Muñoz, A., 2017), was designed. Using this tool, different active attacks were implemented (brute force attacks and DoS), and Internet (IPv4) was monitored from May 2016 until January 2017. The tool uses IP addresses of interest obtained from the search engine Shodan (Shodan, 2017) and scanning tools such as Zmap (Zmap, 2017) or Masscan (Masscan, 2017). These scanning tools have the capability of scanning the whole Internet within few minutes. If further information is required, we strongly recommend reading the user guide included with the tool (Hernández, M; Muñoz, A., 2017).

In the case of Neo4J, more than 1275 databases were found, 298 without any authentication. In the databases without authentication, all the privileges could be used for the implementation of different types of attacks. For example, a denial of service for both Neo4j and the whole machine containing it (Neo4J does not manage neither the RAM memory nor the disk storage efficiently). This problem was even more significant when the machine itself offered a web service through port 80, most of the cases corporative webs.

For the cases without authentication in Neo4J (e.g. Figure 3 and Figure 4), it was possible to obtain the specific version used (v1.9/17, v2.x/251 and v3.0/30).

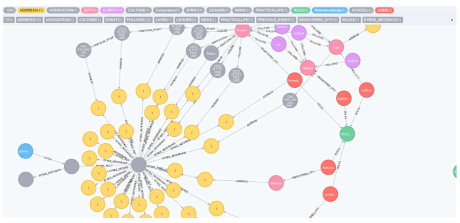


Figure 3: Example of information contained in Neo4j.

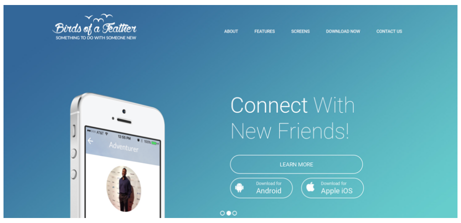
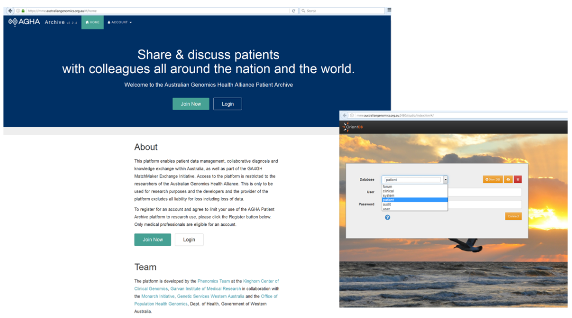


Figure 4: Example of a social network based on Neo4j.

This information makes much easier choosing an optimal type of attack. For example, in 17 cases could be used a public exploit to obtain full control over the machine. A relevant fact was the 5:1 ratio between the *Community* and *Enterprise* licenses. It is important to keep in mind that this fact does have security implications, because the *Community* license does not activate neither logs nor monitor of the *Cypher* queries.

For the OrientDB case, 214 servers were found exposed, among them 553 databases. An analysis was run with the aim of evaluating how often the default credentials were enough, and in 187 the result was positive. Besides, it appears the default database 104 times, that could allow the introduction of queries that provokes DoS attacks or any other kind of misuse of the database (hosting illegal information, cover channel, etc.).


Share & discuss patients
with colleagues all around the nation and the world.

Welcome to the Australian Genomics Health Alliance Patient Archive

About

This platform enables patient data management, collaborative diagnosis and knowledge exchange within Australia, as well as part of the GA4GH MatchMaker Exchange Initiative. Access to the platform is restricted to the researchers of the Australian Genomics Health Alliance. This is only to be used for research purposes and the developers and the provider of the platform excludes all liability for loss including loss of data.

To register for an account and agree to limit your use of the AGHA Patient Archive platform to research use, please click the Register button below. Only medical professionals are eligible for an account.

Team

The platform is developed by the Phenomics Team at the Kinghorn Center of Clinical Genomics, Garvan Institute of Medical Research in collaboration with the Monarch Initiative, Genetic Services Western Australia and the Office of Population Health Genomics, Dept. of Health, Government of Western Australia.

Database: patient
forum
clinical
system
patient
audit
user

Figure 5: Listing of databases in OrientDB associated with a web.

Altogether, 42 servers are vulnerable to the *CVEs* mentioned in the configuration. There was no authentication or the default authentication was activated, the format of the existing files was accessible and in several cases this information could be linked to specific websites in the same IP address, arising the identity of the companies. Regarding information leaks, the implications of this fact are obvious.

During the monitor of these databases was possible to obtain information from more than 600 databases with different sensitivity levels, some of them closely related with public institutions in Spain. In general, sensitive corporative information was found: information about commercial products, hundreds of thousands personal dates (names, e-mails, telephone numbers, etc.), login-passwords, etc. All the discovered information was conveyed to the Guardia Civil's group of telematic crimes (gdt, 2017) and to specific CERTs, who helped to fix and minimize attacks to the vulnerable devices.

6 CONCLUSION

Graph databases are an incipient technology. Nevertheless, they are having an increasing impact on companies and on the management of diverse information, especially in the context of cybersecurity. Hence, and seeking to minimize possible attacks in the future as a consequence of the missuse of these technologies, during this research the configuration and security components implemented within these databases were investigated.

Regarding Neo4J, the most spread database, bad configuration habits of the database are observed, making them vulnerable on the Internet. Measures to prevent force brute attacks are not enough, the key management can be improved and there is not advanced counteractions against attacks that inject *bad queries* (DoS, ransomware, etc.). In this sense, the decision of not implementing key security features, such as logs or monitor in the *Comunnity* version is not the best approach. This evidence was observed on the exposed databases.

In the case of OrientDb there are security problems due to the default configuration. When the server initiates, the server exposes on the Internet the databases by default. Moreover, three users are always created by default, and they have to be removed to prevent improper access. In the same way implements deficient protection measures against brute force attacks and the detection of the version or databases. The security problems discovered forces to release a new version of the analysed graph databases, Neo4j and OrientDB. This is the best recommendation to remedy the discovered security problems.

As a conclusion, excluding the new issues detected, it has to be enhanced how the same design mistakes and dangerous default configurations are systematically carry out. Once again, it is necessary keep in mind the need of audit the emergent technologies for dismissing an eventual negative impact on our organizations. The graph databases are proof of it.

REFERENCES

AuthOrientDB (2017). List databases on orientdb. http://orientdb.com/docs/2.1/Console-Command-List-Databases.html.

Basic Auth (2017). Basic authentication. https://en.wikipedia.org/wiki/Basic_access_authentication. last access 05/22/2017.

Burt, K. (2015). Pkcs #5: Password-based cryptography specification version 2.0. https://tools.ietf.org/html/rfc2898.

CVE-Neo4j (2013). Cve-2013-7259. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-7259. last access 05/22/2017.

Cypher (2017). Intro to cypher. https://neo4j.com/developer/cypher-query-language/. last access 05/22/2017.

Cypher T (2017). Tuning your cypher: Tips and tricks for more effective queries. https://neo4j.com/blog/tuning-cypher-queries/. last access 05/22/2017.

Db-engines (2017). Ranking of graph dbmd 2017. http://db-engines.com/en/ranking/graph+dbms. last access 05/22/2017.

Euler, L. (1736). Solutio problematis ad geometriam situs pertinentis. http://eulerarchive.maa.org/pages/E053.html.

gdt (2017). Gdt. https://www.gdt.guardiacivil.es/. last access 05/22/2017.

Godsil, C; Royle, G. (2001). Algebraic graph theory.

Hammer, C. (1981). Higher-order homophonic ciphers. http://dx.doi.org/10.1080/0161-118191856075.

Hernández, M; Muñoz, A. (2017). Tool grafscan. https://github.com/grafscan/GraFScaN.

Masscan (2017). Port scanner masscan. https://github.com/robertdavidgraham/masscan. last access 05/22/2017.

Mehmood, S. (3 May 2016). Enterprise survival guide for ransomware attacks. https://www.sans.org/readingroom/whitepapers/incident/enterprise-survival-guide-ransomwareattacks-36962.

Neo4j (2013). Neo4j 1.9. https://neo4j.com/release-notes/neo4j-1-9/. last access 05/22/2017.

Neo4j 2.0 (2013). Neo4j 2.0. https://neo4j.com/release-notes/neo4j-2-0/. last access 05/22/2017.

Neo4j 3.0 (2016). Neo4j 3.0. https://neo4j.com/whats-new-in-neo4j-3-0/. last access 05/22/2017.

Neo4j conf (2017). Https configuration. https://github.com/neo4j/neo4j/commit/c9030e5a2efb2c3f43c0f69bebd67ae67e54a286. last access 05/22/2017.

Neo4j connector (2017). Neo4j connector. https://github.com/se38/Neo4a. last access 05/22/2017.

Neo4j lic (2017). Neo4j licenses. https://neo4j.com/licensing/. last access 05/22/2017.

Orientdb (2017). Tutorial about graph-model. http://orientdb.com/docs/last/Tutorial-Document-and-graph-model.html. last access 05/22/2017.

OrientDB expo (2017). Orientdb server. http://orientdb.com/docs/2.2/DB-Server.html.

OrientDB rol (2017). Orientdb security. http://orientdb.com/docs/2.0/orientdb.wiki/Security.html.

Pauli, D (2017). Mongodb ransom attacks soar, body count hits 27,000 in hours. https://www.theregister.co.uk/2017/01/09/mongodb/.

Shodan (2017). The search engine for the web. https://www.shodan.io/. last access 05/22/2017.

Todd, H. (13 de junio de 2009). Neo4j - a graph database that kicks buttox. http://highscalability.com/neo4j-graph-database-kicks-buttox.

Zachary, D. (2015). Big data: Astronomical or genomical? http://journals.plos.org/plosbiology/article/file?id=10.1371.

Zmap (2017). Port scanner zmap. https://zmap.io/. last access 05/22/2017.