NOSQL DATABASES: FORENSIC ATTRIBUTION IMPLICATIONS

W. K. Hauger* **and M. S. Olivier†**

* *ICSA Research Group, Department of Computer Science, Corner of University Road and Lynnwood Road, University of Pretoria, Pretoria 0002, South Africa E-mail: whauger@gmail.com*

† *Department of Computer Science, Corner of University Road and Lynnwood Road, University of Pretoria, Pretoria 0002, South Africa E-mail: molivier@cs.up.ac.za*

Abstract: NoSQL databases have gained a lot of popularity over the last few years. They are now used in many new system implementations that work with vast amounts of data. Such data will typically also include sensitive information that needs to be secured. NoSQL databases are also underlying a number of cloud implementations which are increasingly being used to store sensitive information by various organisations. This has made NoSQL databases a new target for hackers and other state sponsored actors. Forensic examinations of compromised systems will need to be conducted to determine what exactly transpired and who was responsible. This paper examines specifically if NoSQL databases have security features that leave relevant traces so that accurate forensic attribution can be conducted. The seeming lack of default security measures such as access control and logging has prompted this examination. A survey into the top ranked NoSQL databases was conducted to establish what authentication and authorisation features are available. Additionally the provided logging mechanisms were also examined since access control without any auditing would not aid forensic attribution tremendously. Some of the surveyed NoSQL databases do not provide adequate access control mechanisms and logging features that leave relevant traces to allow forensic attribution to be done using those. The other surveyed NoSQL databases did provide adequate mechanisms and logging traces for forensic attribution, but they are not enabled or configured by default. This means that in many cases they might not be available, leading to insufficient information to perform accurate forensic attribution even on those databases.

Key words: database forensics, forensic attribution, NoSQL, survey.

1. INTRODUCTION

In recent years NoSQL databases have gained popularity both with developers who build new systems, and within organisations who want to optimise and improve their businesses [1]. Both of those parties are trying to adapt their information systems to meet today's data demands.

Certain NoSQL databases have even moved up from being niche products to leaders in Gartners Magic Quadrant for Operational Database Management Systems [2]. Gartner considers databases in the leaders quadrant to be of operational quality. According to Gartner, leaders generally represent the lowest risk for customers in the areas of performance, scalability, reliability and support.

The forerunners of today's NoSQL databases were started by big web companies such as Google, Amazon and Facebook to help them build and support their businesses [3]. After they made these new databases public and open source, other big web companies such as Twitter, Instagram and Apple started to use them as well [3]. This has lead to the development of a number of NoSQL databases based on the ideas and models of the original databases.

The use of NoSQL databases has started to filter down to ordinary organisations who are now also starting to use NoSQL databases for various purposes in their business processes. The consequence of this is that more and more data is being placed in NoSQL databases. This includes private and sensitive information which has to be kept secure and confidential.

Additionally one big area of use for NoSQL is Big Data. As the name implies, Big Data deals with vast amounts of data that needs to be stored, analysed and retrieved. Copious amounts of this data are normally unstructured and make NoSQL databases such an attractive proposition. However, this also means that unauthorised access to such NoSQL databases has the potential to expose very large amounts of information.

Together with the rise in popularity, the increased storage of data has made these NoSQL databases attractive targets for hackers, state actors, extortionists etc. Data security has thus started to become an important aspect of NoSQL databases. Some research has already been conducted into the security features provided by these NoSQL database and found them to be lacking [4, 5].

In contrast to the previous research, this paper looks at NoSQL database security from a forensic perspective. The focus is in particular on forensic attribution. Performing forensic attribution in digital systems is difficult and inherently limited [6]. These limitations include attribution delay, failed attribution, and mis-attribution.

This is because on the one end the actions that need to be attributed occurred in the digital world, but on the other end the actors that are ultimately responsible are located in the physical world. Therefore various researchers have

Based on: "Forensic attribution in NoSQL databases", by W. Hauger and M. Olivier which appeared in the Proceedings of Information Security South African (ISSA) 2017, Johannesburg, 16 & 17 August 2017.

proposed different levels, categories or steps of attribution that can be performed. These different levels, categories or steps not only have increasing degrees of difficulty and complexity, but they also extend different distances from the actions in the digital world to the responsible actors in the physical world. A more detailed discussion follows in section 3.

Always performing the full spectrum of attribution from the actions to the actors might not even be necessary. Clark and Landau argue that the occasions when attribution to the level of an individual person is useful are actually very limited [7]. They note that, although criminal retribution requires identifying a specific person and assigning blame, "the evidence that is finally brought into court is unlikely to be [a] 'forensic quality' computer-based identity, but rather other sorts of physical evidence found during the investigation" [7].

Keeping the previous statement in mind, a valuable first step would be to tie the actions in question to a program or process on a machine inside the digital realm [8]. However, even tying actions to processes can be difficult without enough information sets that can be correlated to form a consistent chain of events [8]. Relational databases provide one such set of information in the form of traces in various log files and in system tables [9]. This information can then be used in conjunction with other information sets from outside the database to help perform attribution of the actions that occurred inside the database.

These traces can be left by measures such as access control and logging/auditing which are normally part of the security model of all relational databases. Consequently, this paper scrutinises the security features that are available in NoSQL databases and how useful their traces can be for forensic attribution. A survey of these specific security measures in NoSQL databases was conducted to determine what information they can provide to aid with forensic attribution in the case of a forensic examination.

There are more than two hundred NoSQL database implementations available [10] and to examine the security measures of all those databases would be prohibitive. Many of those databases are still experimental or being used in low volumes. Thus only a few NoSQL database management systems (DBMSs) were chosen of which the access control and logging features were studied. The choice was based on popularity and to be representative of the main types of data models used by NoSQL databases. Section 2 provides more details about the main NoSQL data types.

The NoSQL DBMSs examined were MongoDB, Cassandra, Redis and Neo4j. These selected NoSQL databases are among the most popular based on the number of web pages on the Internet according to DB-Engines ranking method [11]. They are being adopted in various markets in the industry and their prominence in those markets means that they would be encountered fairly often by the general digital forensic investigator.

To study the security features, the official documentation of the latest version of the selected NoSQL DBMS as found published on the manufacturer's website was used [12–15]. At the time of the examination the latest versions available were as follows: MongoDB 3.4, Cassandra 3.10, Redis 3.2 and Neo4j 3.1.3.

Even though each one of the selected NoSQL databases support scaling and data distribution via multi-node configurations, these databases were only considered as single node installations. Thus a discussion on distributed log files and the added complexities falls outside the scope of this study.

The remainder of this paper is structured as follows: Section 2 first gives a general introduction to NoSQL databases and their characteristics. Then it reviews each of the chosen NoSQL databases. Section 3 provides an overview of the field of forensic attribution. Section 4 then surveys the selected NoSQL databases regarding authentication, authorisation and logging. Section 5 analyses the results of the survey. Section 6 discusses the implications on digital forensics and specifically forensic attribution. Section 7 concludes this paper and contemplates future research.

2. NOSQL DATABASES

This section first provides a general introduction to NoSQL databases. Then the NoSQL databases chosen for the survey are examined in more detail.

2.1 NoSQL databases and types

The NoSQL movement was the development of new types of databases that were not relational and did not use the structured query language (SQL) as the data access language. These new database types were being created to address new demands in data forms and size that seemingly could no longer be met by existing relational databases and SQL.

NoSQL databases have more flexible data structures that can be completely schemaless. This allows for easier storage of unstructured and heterogeneous data. They also provide easier horizontal scalability to cater for big sets of data and data that grows unpredictably.

The "NoSQL" moniker became popular when Eric Evans chose it as the name for an event that Johan Oskarsson organised to discuss open source distributed databases [16]. Eric felt that the whole point of the event was to seek out alternatives that one needed to solve a problem that relational databases were a bad fit for. The event was the beginning of a movement that grouped together all database projects that were not relational.

Some people have objected to the NoSQL term for these new databases [17, 18], because it sounded like a definition based on what these databases were not doing rather than what they were. In recent years it has been suggested that

the NoSQL term be changed from meaning "No SQL" to "Not Only SQL". This is to express that NoSQL no longer meant anti-SQL and anti-relational, but rather expressed the notion that other database types besides relational ones existed that could help address the new data types and storage demands of today's information society.

Today, a number of distinct types of NoSQL databases have established themselves. To make this research inclusive, databases from all the main types should be included in the study. The availability and usability of features such as access control and logging could also be influenced by how the different databases function internally. It might not be simply a matter of implementation to provide these features, but feasibility and practicality might also play a role.

It is thus worthwhile to take a closer look at the details of each of those NoSQL database types. The main types of NoSQL databases are the following four types: Document databases or stores, Key-value pair databases or stores, Column family store databases or wide column stores and Graph databases or stores [19]. A short summary of each type now follows.

Document Stores:

Document databases, also known as document stores or document-oriented databases, use a document-oriented model to store data. They store a record and its associated data within a single data structure called a document. Each document contains a number of attributes and associated values. Documents can be retrieved based on attribute values using various application programming interfaces (APIs) or query languages provided by the DBMS [19].

Document stores are characterized by their schema-free organization of data. That means that records do not need to have a uniform structure, i.e. different records may have different attributes. The types of the values of individual attributes can be different for each record. Records can also have a nested structure, while attributes can have more than one value such as an array.

Document stores typically use standard formats such as JavaScript Object Notation (JSON) or Extensible Markup Language (XML) to store the records [19]. This then allows the records to be processed directly in applications. Individual documents are stored and retrieved by means of an key. Furthermore, document stores rely on indexes to facilitate access to documents based on their attributes [20].

Wide Column Stores:

Wide column stores, also called extensible record stores, store data in records with an ability to hold very large numbers of dynamic columns. A column is the basic unit of storage and consists of a name and a value [19].

Any number of columns can be combined into a super column, which gives a name to a sorted set of columns. Columns are stored in rows, and when a row contains columns only, it is known as a column family. When a row contains super columns, it is known as a super column family [20].

As in document stores, column family databases do not require a predefined fixed schema. Different rows can have different sets of columns and super columns. Since the column names as well as the record keys are not fixed, and a record can have millions of columns, wide column stores can be seen as two-dimensional key-value stores [19].

Key-Value Stores:

Key-value stores are probably the simplest form of databases. They can only store pairs of keys and values, as well as retrieve values when the key or identifier is known. These systems can hold structured or unstructured data.

A namespace is a collection of identifiers. Keys must be unique within a namespace. A namespace could correspond to an entire database, which means all keys in the database must be unique. Some key-value stores provide for different namespaces within a database. This is done by setting up data structures for separate collections of identifiers within a database [19].

These simple systems are normally not adequate for complex applications. On the other hand, it is exactly this simplicity, that makes such systems attractive in certain circumstances. For example resource-efficient key-value stores are often applied in embedded systems or as high performance in-process databases.

One of the earliest such embedded key-value databases is Berkeley DB which was first released in 1991. It was developed at the University of California, Berkeley to replace certain patented components in their Unix release BSD 4.3. In 1992 BSD 4.4 was released which included Berkeley DB 1.85 [21].

Graph Stores:

Graph stores also known as graph databases are DBMSs with Create, Read, Update, and Delete (CRUD) methods that expose a graph data model. A graph database represents data in structures called nodes and relationships. A node is an object that has an identifier and a set of attributes. A relationship is a link between two nodes that contain attributes about that relation [19].

Some graph databases use native graph storage, which is designed to store and manage graphs directly. Other graph databases serialize the graph data into relational or object-oriented databases, or use other types of NoSQL stores [20]. In addition to having a certain approach to storing and processing graph data, a graph database will also use a specific graph data model. There are several

different graph data models commonly used which include property graphs, hypergraphs, and triples.

Graph databases dont depend so much on indexes because the graph itself provides a natural index. In a graph database using native graph storage, the relationships attached to a node provide a direct connection to other related nodes. Graph queries use this characteristic to traverse through the graph [20]. Such operations can be carried out very efficiently, typically traversing millions of nodes per second. In contrast, joining data through a global index can be many orders of magnitude slower.

2.2 Surveyed NoSQL databases

Currently, the top five ranked NoSQL databases according to DB-Engines DBMS ranking are in order: MongoDB (document store), Cassandra (wide column store), Redis (key-value store), HBase (wide column store) and Neo4j (graph store) [11]. The top five thus represent all four NoSQL database types introduced above. To eliminate any possible bias of the survey due to multiple databases of the same type, the second wide column store HBase was excluded. More details about each of the other four databases now follow.

MongoDB:

MongoDB is an open-source document database that provides high performance, high availability, a rich query language and automatic scaling. It is published under a combination of the GNU Affero General Public License (AGPL) and the Apache License. The name MongoDB is derived from ”humongous database”, which alludes to the huge size a MongoDB database can have.

The software company 10gen began developing MongoDB in 2007 as a component of a planned platform as a service product. In 2009, the company shifted to an open source development model, with the company offering commercial support and other services. In 2013, 10gen embraced the database it had created and changed its name to MongoDB Inc. [22].

A record in MongoDB is a document, which is a data structure composed of a number of field and value pairs. MongoDB documents are similar to JSON objects. The values of fields may include other documents, arrays, and arrays of documents. MongoDB lists the advantages of using documents as follows: Firstly documents (and by extension objects) correspond to native data types in many programming languages. Secondly, embedded documents and arrays reduce the need for expensive joins. And finally dynamic schemas support fluent polymorphism [23].

MongoDB provides high performance data persistence and access through the use of the following features: Firstly it supports embedded data models which reduce I/O activity on the database system. Secondly it’s indexes can include keys from embedded documents and arrays thereby supporting faster queries. MongoDB uses B-trees for both data and index persistence. MongoDB has a rich query language that supports create, read, update and write (CRUD) operations as well as data aggregation, text search and geo-spatial queries [23].

MongoDB uses a replication facility called a “replica set” to provide automatic failover and data redundancy. A replica set is a group of MongoDB servers that maintain the same data set. Furthermore, MongoDB provides horizontal scalability by using sharding, which distributes data across a cluster of machines. It also supports the creation of zones of data based on a shard key. In a balanced cluster, MongoDB will direct reads and writes covered by a zone only to those shards inside the zone [23].

Prominent users of MongoDB include Metlife, Expedia, Ebay, SAP, SAGE, KPMG and Forbes [24].

Cassandra:

Cassandra is a free and open-source distributed wide column store DBMS. It is an Apache project published under the Apache 2.0 license. Cassandra is designed to handle large amounts of data across many commodity servers thereby providing high availability with no single point of failure. Cassandra offers support for clusters either in a single datacenter or spanning across multiple datacenters, with asynchronous masterless replication.

Avinash Lakshman and Prashant Malik initially developed Cassandra at Facebook to power the Facebook inbox search feature. They named their database after the Trojan mythological prophet Cassandra, who was given the power of prophecy by Apollo in order to seduce her. When she refused him favours, he cursed her prophecies to be never believed. The name thus alludes to a cursed oracle.

Facebook released Cassandra as an open-source project on Google code in July 2008 [25]. In March 2009 it became an Apache Incubator project and graduated to a top-level Apache project in February 2010. Cassandra is written in Java and thus available on any platform that can provide a Java virtual machine (JVM).

Cassandra’s column family (also called table) resembles a table in a relational database. Column families contain rows and columns. Each row is uniquely identified by a row key. Each row has multiple columns, each of which has a name, value, and a timestamp. Unlike a table in a relational database, different rows in the same column family do not have to share the same set of columns, and a column can be added to one or multiple rows at any time without blocking updates or queries.

Cassandra Query Language (CQL) is the primary interface into the Cassandra DBMS [26]. Using CQL is similar to using SQL. CQL and SQL share the same abstract idea of a table constructed of columns and rows. The main difference from SQL is that Cassandra does not support joins or sub-queries. Instead, Cassandra emphasizes

denormalization through CQL features like collections and clustering specified at the schema level.

Cassandra uses a combination of memory tables (Memtables) and sorted string tables (SSTables) for persistence. Memtables are in-memory structures where Cassandra buffers all of its writes. When the Memtables are full, they are flushed onto disk by sequentially writing to the SSTables in append mode. Once written, the SSTables become immutable. Using this approach make it possible for Cassandra to avoid having to read before writing. Reading data involves combining the immutable sequentially-written SSTables to retrieve the correct query result [26].

In Cassandra, data is automatically replicated to multiple homogeneous nodes for fault-tolerance. A replication strategy determines the nodes where the replicas are placed. Cassandra employs a peer-to-peer distributed system across the nodes whereby the data is distributed among all nodes in the cluster [26]. Failed nodes in a cluster can be replaced with no downtime.

Prominent users of Cassandra include CERN, Netflix, Reddit and eBay [27].

Redis:

Redis is an open source key-value store that is published under the Berkeley Software Distribution (BSD) license. The in-memory data structure store can be used as a database, a cache or a message broker. The name Redis stands for REmote DIctionary Server. Redis was developed by Salvatore Sanfilippo who released the first version in May 2009. He was hired by VMware in March 2010 to work full time on Redis [28]. In 2015, Salvatore Sanfilippo joined Redis Labs which now sponsors development.

Redis maps keys to different types of values. It not only supports simple data structures such as strings but also abstract data structures such as hashes, lists, sets and sorted sets. Geo-spatial data is now supported through the implementation of the geohash technique.

The type of a value determines what operations (called commands) are available for the value itself. Redis supports high-level, atomic, server-side operations like appending to a string, incrementing the value in a hash, pushing an element to a list, computing set intersection, union and difference and sorting of lists, sets and sorted sets [29].

Redis is written in ANSI C and works in most POSIX systems like Linux, various BSD operating systems and OS X without external dependencies. It works with an in-memory dataset which can be optionally be persisted by either by dumping the dataset to disk every once in a while, or by appending each command to a log file.

Redis has built-in replication using a master-slave configuration which can be performed either via the dump file or directly from process to process. Redis also supports memory eviction methods such as Least Recently Used (LRU) which allows it to be used as a fixed size cache. Additionally Redis supports the publish/subscribe messaging paradigm, which allows it to be used a messaging platform.

Redis has a built-in Lua interpreter which can be used to write complex functions that run in the Redis server itself. Lua is a lightweight multi-paradigm programming language. Add-on Redis products provide additional features such as high availability via Redis Sentinel and automatic partitioning with Redis Cluster.

Prominent users of Redis include Twitter, Pinterest and Flickr [30].

Neo4j:

Neo4j is an open-source graph database management system that is an ACID-compliant transactional database with native graph storage and processing [31]. It is published under dual licence of the GNU Affero General Public License (AGPL) and the GNU Public License (GPLv3).

Neo4j is developed by Neo Technology, Inc. which released the first version in February 2010. It is implemented in Java and accessible from software written in other languages using the Cypher Query Language (CQL) through a transactional HTTP endpoint, or through the binary "bolt" protocol (Not related to the Cassandra Query Language).

The main features and capabilities of CQL are as follows: Firstly it works by matching patterns of nodes and relationships in the graph, to extract information or modify the data. Secondly it has the concept of variables which denote named, bound elements and parameters. Thirdly it can create, update, and remove nodes, relationships, labels, and properties. Lastly it is used to manage indexes and constraints [32].

In Neo4j, everything is stored in the form of either an edge, a node, or an attribute. Each node and edge can have any number of attributes. Both the nodes and edges can be labelled. Labels can be used to narrow searches. Neo4j uses on-disk linked lists for persistence.

Joining data together in Neo4j is performed as navigation from one node to another which provides linear performance degradation compared to relational databases, where the performance degradation is exponential for an increasing number of relationships [31].

Prominent users of Neo4j include Walmart, Monsanto and Ebay [33].

3. ATTRIBUTION

Having completed the introduction of NoSQL databases and the presentation of the selected NoSQL databases, focus is now placed on attribution. First an overview of the field is given to provide context for the survey. Then forensic attribution processes and techniques are discussed and their implications on database forensics explored.

3.1 General attribution

The Oxford English dictionary defines the term *attribution* as follows: "The action of regarding something as being caused by a person or thing". Attribution is performed in a number of diverse application areas. These include clinical psychology attribution, nuclear attribution, authorship attribution and cyber attack attribution.

In clinical psychology, attribution refers to the process by which individuals explain the causes of behaviour and events [34]. In nuclear forensics, nuclear attribution is the process of tracing the source of nuclear material from a radiological incident whether accidental (e.g. nuclear waste spill) or intentional (e.g. nuclear explosion) [35]. Authorship attribution refers to the process of inferring characteristics of the author from the characteristics of documents written by that author [36].

Cyber attack attribution has been defined and researched by various authors. Wheeler and Larson in their paper for the U.S. Department of Defense (DoD) defined it as "determining the identity or location of an attacker or an attackers intermediary" [6]. They define the resulting identity as a persons name, an account, an alias, or similar information associated with a person. A location is interpreted as a physical (geographic) location, or a virtual location such as an IP address or MAC address.

Boebert breaks the attribution question down into two attribution problems: technical attribution and human attribution [37]. According to the author, technical attribution consists of analysing malicious functionality and packets, and using the results of the analysis to locate the node which initiated, or is controlling, the attack. Human attribution on the other hand, consists of taking the results of technical attribution and combining it with other information to identify the person or organization responsible for the attack.

Clark and Landau in their paper "Untangling attribution" contend that there are many types of attribution and different types are useful in different contexts [7]. For example, attribution on the Internet could mean the identification of the owner of the machine (e.g. the company or organisation), the physical location of the machine (e.g. city or country) or the individual who is actually responsible for the actions.

Clark and Landau also define three classes of attacks: bot-net based attacks (e.g. DDoS), identity theft and data ex-filtration and espionage. Based on these classes different attribution types and techniques are more suitable or applicable than others. The timing of when attribution is performed also plays an important role.

For example during a DDoS attack, mitigation might be the most immediate concern. Attribution is then needed to identify the machines launching the attack so that they can be blocked. However, after the DDos attack is over, focus may shift towards retribution as deterrence. Attribution is then needed to identify the actors responsible so that they can be prosecuted [7].

3.2 Forensic attribution

When attribution is done as part of an investigation using scientific methods, the term forensic attribution is used. Forensic attribution is performed during the digital evidence interpretation step in a forensic examination [38]. This step is part of the investigative processes as defined in the ISO/IEC 27043 standard that describes incident investigation principles and processes [38].

Forensic attribution processes:

As already touched on in the introduction, a number of researchers have proposed different forensic attribution processes. These diverse processes define different levels, categories or steps of attribution that can be performed.

Cohen for example sees end-to-end attribution made up of four different levels of attribution [39]. The first two levels are performed in the digital world. Level 1 attempts to identify the closest computer involved while level 2 tries to pinpoint the source computer that initiated the actions. However, the next two levels of attribution are performed in the physical world. Level 3 attempts to identify the individual that caused source computer to act as it did, while at level 4 the organisation behind the individual is being sought.

Shamsi et al propose three steps of identification for attribution [40]. The first step deals with the identification of the cyber weapon used to launch the actions. They use the definition of *cyber weapon* given by Rid and McBurney. They, in turn, define cyber weapon as "computer code that is used, or designed to be used, with the aim of threatening or causing physical, functional, or mental harm to structures, systems, or living beings" [41].

Thus step 1 is executed in the digital realm. Step 2 deals with the identification of the country or city of the actor, while step 3 addresses the identification of the actor whether it is a person or an organisation. The last two steps thus take place in the physical world. Similarly, Clark and Landau define three categories into which attribution can fall [7]. These categories are the machine, the person, and the aggregate identity, such as a state actor. Again, the first category of attribution is executed in the digital realm, while the other two happen in the physical world.

Table 1 summarises these different attribution processes.

Table 1: Attribution Processes Summary

Author(s)	Digital Realm	Physical Realm
Cohen [39]	• Identify closest computer • Identify initiating computer	• Identify individual behind initiating computer • Identify organisation behind individual
Shamsi et al. [40]	• Identify cyber weapon	• Identify country/city • Identify person/organisation
Clark and Landau [7]	• Identify computer	• Identify individual • Identify organisation

Even though the authors use different terminology to describe the parts of their processes (steps/levels/categories), the parts and their goals seem to be very similar. For the sake of clarity this paper is going to overlook the deeper meaning of the authors' chosen terminology and from here on simply refer to the different parts as steps.

In the digital realm the steps from all authors have the same goal: to identify the computing device(s) responsible for the actions that are being examined. Various digital forensic attribution techniques can be used to achieve this goal. The more difficult steps are performed in the physical realm. They attempt to identify the individuals or actors responsible for the actions that are being examined. As already indicated in the introduction, this kind of attribution may not be always be needed. The paper will therefore concentrate on attribution steps that are performed in the digital realm.

Forensic attribution techniques:

There are a large number of digital attribution techniques with each technique having certain strengths and weaknesses. A taxonomy of these attribution techniques is provided by Wheeler and Larson [6]. According to them no single technique can replace all others and a combination of techniques can help compensate for their respective weaknesses.

One of the techniques that the forensic examiner can employ is to make inferences based on authentication and authorisation information [8]. This information enables the forensic examiner to create a basis for attribution. The authentication information provides the actors of a digital system and the authorisation information the actions that these actors can perform [42].

Another technique is to order and connect the different traces found in digital systems to build a chain of events [8]. The sequences of these events describe how the system arrived at the current state. By determining the actions that lead to the events and the actors that performed the actions, the person or program responsible can possibly be identified [42].

Performing forensic attribution in relational databases was investigated by Olivier [9]. He showed that database forensics can use the same techniques as general digital forensics to perform attribution. The traces in a relational database are available in various log files and also stored inside system tables. Furthermore, the authentication of database users and the authorisation of their operations has been standardised and is built into many relational databases [43].

4. NOSQL SECURITY SURVEY

A survey of the documentation for the chosen four NOSQL databases was conducted. Comparable information was obtained on the availability of the following features: authentication, authorisation and logging. The official documentation was used as far as possible, but in some cases document holes were supplemented with additional sources. In some cases the information was readily available, while in other cases it was necessary to delve into the database software files. This section presents the results of this survey.

It should be noted, that only the features available in the official free and community editions of the selected NoSQL databases were analysed. Some of the NoSQL databases also have paid-for enterprise editions available that provide additional features (See for example Neo4j Editions [44]). These additional features include enhancements and additions to authentication, authorisation and logging.

The results for the three features are presented in the same order for all the selected databases. This is to enable direct comparison between the different NoSQL databases and allow commonalities and/or differences to be established for later discussion.

4.1 MongoDB

Unless otherwise indicated, this section uses the official MongoDB documentation [12] to paraphrase the relevant information to indicate the availability of the surveyed features.

MongoDB supports a number of authentication mechanisms that clients can use to verify their identity. These include SCRAM-SHA-1 and x.509 client certificates. SCRAM-SHA-1 is an authentication mechanism from the Salted Challenge Response Authentication (SCRAM) family that uses the SHA-1 hash function. It is a mechanism for authenticating users with passwords and defined in the IETF standard RFC 5802 [45].

MongoDB employs role-based access control to govern access to a MongoDB system. A user is conferred one

or more roles that determine the user's access to database resources and operations. Outside of role assignments, the user has no access to the system. MongoDB does not enable access control by default, but it can be enabled via the configuration file or a start-up parameter.

Since MongoDB does not have a built-in default user, an appropriate administration user must be created before authentication is enabled. Alternatively, MongoDB provides an exception where it allows an unauthenticated connection on the local loopback interface to the admin database. Once an appropriate administration user has been created via this connection, no further actions can be performed and this connection needs to be terminated to establish a new authenticated one.

MongoDB provides a number of built-in roles that can be used to control access to a MongoDB system. Each of these roles have specific privileges assigned to them. The roles are divided into different categories such as database user, database administrator, superuser etc. However, if the specific privileges of the built-in roles are not sufficient, one can create new roles with the desired privileges in a particular database.

A role grants privileges to perform sets of actions on defined resources. A given role applies to the database on which it is defined. Access can be granted on a whole cluster, a specific database in the cluster or to individual collections inside a database. Privileged actions that are available to roles are grouped together as follows: query and write actions, database management actions, deployment actions, replication actions, sharding actions and server administration actions.

MongoDB database instances can report on all their server activity and operations. Per default, these messages are written to standard output, but they can be directed to a log file via the configuration file or a start-up parameter. MongoDB's default log verbosity level includes just informational messages. This can be changed to include debug messages by setting the verbosity to a higher level.

Additionally, MongoDB allows logging verbosity to be controlled at a finer grain by providing verbosity settings on a component level. These components include items such as access control, commands, queries etc. Unless explicitly set, each component has the verbosity level of its parent. MongoDB verbosity levels range from the informational default of 0 to the most verbose debug level of 5.

When logging to a file is enabled, MongoDBs standard log rotation approach archives the current log file and starts a new one. This normally occurs when the MongoDB instance is restarted. While the MongoDB instance is running, this can also be triggered by either issuing the "logRotate" command inside the database or by sending the SIGUSR1 signal from the OS to the MongoDB process id.

4.2 Cassandra

Unless otherwise indicated, this section uses the official Cassandra documentation [13] to paraphrase the relevant information to indicate the availability of the surveyed features.

Cassandra provides pluggable authentication that can be configured via settings in the configuration file. The default Cassandra configuration uses the AllowAllAuthenticator which performs no authentication checks and therefore requires no credentials. It is used to disable authentication completely. Cassandra also includes the PasswordAuthenticator, which stores encrypted credentials in a system table. This is used to enable simple username/password authentication.

Cassandra uses a role-based access control framework, but provides no fixed or pre-defined roles. Cassandra roles do have a login property and a superuser property. The default Cassandra user has these properties set so it can be used to setup further users and roles once authentication has been enabled. Users and roles are the exact same concept, but to preserve backward compatibility they are both still used. User statements are simply synonyms of the corresponding role statements.

Cassandra also provides pluggable authorisation that can be configured in the same configuration file as authentication. By default, Cassandra is configured with the AllowAllAuthorizer which performs no checking and so effectively grants all permissions to all roles. This is used if the AllowAllAuthenticator is the configured authenticator. Cassandra also includes the CassandraAuthorizer, which implements full permissions management functionality and stores its data in Cassandra system tables.

Permissions on various resources are granted to the roles. The permissions available depend on the type of resource. Cassandra provides the following resource types: data resources such as keyspaces and tables, function resources, database roles and Java managed beans (MBeans). The resource types are structured as hierarchies and permissions can be granted at any level of these hierarchies and they flow downwards.

Cassandra provides all of the following permissions: CREATE, ALTER, DROP, SELECT, MODIFY, AUTHORIZE, DESCRIBE, EXECUTE. A matrix determines which permissions can be applied to which resources. One can grant individual permissions to resources or use the GRANT ALL syntax to grant all applicable permissions to a resource.

Cassandra uses the Java logging framework Logback to create various log files about everything that occurs in the system. Java logging classifies messages in levels [46], where a lower level of messages will include all the higher level ones as well. For example, the INFO level will include message from the higher ERROR level, while the

lower DEBUG level will include the higher level INFO and ERROR messages. By default the following two log files are created: the system log file which contains all the INFO level messages produced in the system and the debug log file which contains all the DEBUG level messages. The debug log file additionally contains caller information as well.

Another log file available in Cassandra is the commit log. To enhance performance, Cassandra keeps column updates in memory and periodically flushes those changes to disk. To prevent data losses when the system goes down before flushing, these updates are also written to the commit log. When Cassandra starts up again, it reads the commit log back from the last known good point in time and re-applies the changes in the commit log so it can get into the same state as when it went down. Although the commit log only contains the most recent changes that have not been flushed to disk yet, there is a configuration option that will archive the contents of the commit log.

4.3 Redis

Unless otherwise indicated, this section uses the official Redis documentation [14] to paraphrase the relevant information to indicate the availability of the surveyed features.

Redis is an in-memory database that is designed to be run inside trusted environments and accessed by trusted clients. Untrusted access is expected to be mediated by an intermediary layer that implements access control, validates user input and determines what operations may be performed against the Redis database instance.

Although Redis does not implement access control, it does provide a tiny layer of authentication that can be enabled by editing the configuration file and setting a password. When the authentication layer is enabled, Redis will refuse any queries by unauthenticated clients. A client then must authenticate itself by sending the AUTH command followed by the password. The AUTH command, like every other Redis command, is sent unencrypted.

The purpose of the authentication layer is to serve as a protection layer against the accidental exposure of a Redis database instance to external untrusted environments. To force the setting of a password, a Redis instance in default configuration will only start in protected mode. In protected mode the Redis instance only accepts clients on the local loopback interface while throwing errors on all other available interfaces. Once the password has been set, the other configured interfaces will accept client connections.

Redis has no form of authorisation. Once a client is authenticated, any command can be called including the FLUSHALL command which will delete the whole data set. As mitigation, Redis allows commands to be renamed into unguessable names, so that normal clients can be limited to a specified set of commands. Systems that provide and manage Redis instances would then still be able to execute the renamed commands.

Redis does have some form of logging, although it is advised that it be only used for debugging purposes. The Redis “Slow Log” is a system to log queries that exceeded a specified execution time. However, by setting the execution time threshold to zero all commands including queries will be logged. Keeping with its in-memory nature, Redis keeps the slow log in memory. To prevent over usage of memory for logging purposes, by default only the last 1024 slow log entries will be kept. To retrieve the slow log entries, the SLOWLOG GET command needs to be used [47].

Another form of command logging happens when append-only file (AOF) persistence is enabled. When enabled, every time the Redis database instance receives a command that changes the dataset (e.g. SET) it will append it to the AOF. The purpose of the AOF is to rebuilt the state after the database was shutdown without a snapshot of the current state. To prevent the file from growing uncontrollably, Redis can from time to time rewrite the actual stored commands with the shortest sequence of commands needed to rebuild the current dataset in memory.

4.4 Neo4j

Unless otherwise indicated, this section uses the official Neo4j documentation [15] to paraphrase the relevant information to indicate the availability of the surveyed features.

Neo4j provides a basic authentication layer that is enabled by default. It has a built-in default user for whom the password can be set during installation. Should the password not be changed during installation, Neo4j will prompt for a password change on first connection. Additional users can be added to the database by the default user once authenticated.

Neo4j has no form of authorisation. This implies that once a client is authenticated, any operation can be performed on the database. Additionally Neo4j only accepts client connections on the local loopback interface in default configuration. External interfaces for remote connectivity need to be configured explicitly.

Neo4j does provides some logging. Traffic on the HTTP/HTTPS connector is logged to a file called http.log. However, this traffic logging is not enabled by default.

The enterprise version of Neo4j however, does provide a role-based access control framework that furnishes built-in roles as well as the ability to add custom roles. It also provides additional logging capabilities to audit security events and queries executed. These capabilities need to be configured first, since they are not enabled by default.

5. DISCUSSION

In this section the results from the security feature survey in the previous section are discussed using a few summary tables. Section V then discusses the implications of these results on forensic attribution.

Table 2: NoSQL Security Features

Database	Authentication	Authorisation	Logging
MongoDB	Yes	Yes	Yes
Cassandra	Yes	Yes	Yes
Redis	Yes	No	Yes
Neo4j	Yes	No	Yes

Table 2 summarises the results from the survey of access control and logging of the selected NoSQL databases. The first result this summary shows, is that all of the surveyed NoSQL databases do support authentication. However the second result is that two of the NoSQL databases do not provide authorisation. This divides the surveyed NoSQL databases into two groups: The first group of databases control both who can access them and what operations the authenticated users can perform. The second group of databases only control who can access them, but not what the authenticated users can do.

Specifically, Redis only provides a thin authentication layer that does not have different users, but rather restricts client access via a simple password. Since it has no differentiated user access, Redis also does not provide any authorisation. Neo4j also does not provide any role based authorisation, even though differentiated user authentication is supported. This implies that in both those databases all clients have the same full control over all database operations once they have been authenticated.

The third result that the summary in Table 2 shows, is that all of the surveyed NoSQL databases do provide some form of logging. It should be noted that this survey looked at all the log files that were being generated by the chosen NoSQL databases, not only audit logs. Some of the log files that were surveyed, are only created when special features in the database are enabled, while other log files are created by the storage mechanism that the particular database uses. This means that rather than being general log files, these files are specialised log files that contain only specific type of messages.

Some NoSQL databases like MongoDB and Redis include the ability to log queries that took particularly long to complete. In the case of Redis, the threshold used to determine when to log slow queries can be changed to zero, which will make Redis log every query executed. Thus the normal Redis slow log can be turned into a query audit log.

Table 3 summarises the default state of the security features that are available for the surveyed NoSQL databases. This summary shows that only one of the surveyed NoSQL databases comes with access control enabled by default. The implication of this result is that the installations of all those other NoSQL databases will be accessible to anyone without explicit configuration changes.

Table 3: Features Enabled by Default

Database	Access Control	Logging
MongoDB	No	No
Cassandra	No	Yes
Redis	No	Yes
Neo4j	Yes	No

A small security consolation is that some of these NoSQL databases will per default only accept client connections on the local loopback interface. This means that no remote access is possible and only clients on the same machine as the database can connect.

In the case of MongoDB, this default "local loopback only" state is created with the value of the network configuration option, which can easily be changed to the network interface of the machine. This single change will then open up the MongoDB database to remote clients without access control. In the case of Redis, this "local loopback only" state is enforced by a separate configuration option. However, by changing it and the network configuration option, the Redis database can be opened up to remote clients without authentication.

Table 3 also shows that logging is not enabled by default on some databases. So even though for example MongoDB has great logging capabilities that can audit database access and operations, none of that is available by default. Only after careful configuration of the various settings will the same information be available as found in many relational databases.

In the case of Neo4j the fact that logging is not enabled by default is not a great loss. This is because only HTTP traffic logging is available in the community edition of Neo4j. The logging capabilities for security events and queries is only available in the paid-for enterprise edition.

6. FORENSIC IMPLICATIONS

This sections considers the implications of the results from the previous section on forensic examinations and particularly forensic attribution. The availability of access control and logging/auditing in the surveyed NoSQL databases is considered separately.

6.1 Access Control

The traces or artefacts from access control in a database can help the forensic examiner as follows: firstly, the

authentication traces can provide a list of users that connected around the time the operations being examined were performed. Secondly, the authorisation matrix can narrow down this list based on who was authorised to perform the operations in question. The first group of NoSQL databases that was identified in the previous section can aid forensic attribution in this way.

The second group of NoSQL databases that the survey identified, only have authentication available, but no authorisation. The implication is that in those databases all clients have the same full control over all database actions once they have been authenticated. This means it will not be possible to narrow down the list of users based on the operations they are authorised to perform, since theoretically all of the users had the authority to perform the operations being examined.

One of the databases in the second group also has no concept of a database user and just provides simple password based access. From a security standpoint this simple authentication layer provides an improvement over having no authentication, but from an forensic attribution standpoint, it adds almost no additional value. The forensic examiner can only deduce that the responsible person was in possession of the correct password, provided the security model of the database is sound and no unauthenticated access is possible.

The survey also showed that none of the selected NoSQL databases have authentication enabled by default. The forensic examiner is thus dependent on the database administrator to have enabled authentication for possible access control traces. But without these access control traces being persisted into a log file or some other data file, the mere presence of access control in the database is not sufficient to aid forensic attribution.

6.2 Logging

The different log files that were encountered during the survey of the selected NoSQL databases can be divided into three groups: audit logs, system logs and storage logs.

Audit Logs: Audit logs maintain a record of various activities in the database for later review or possible debugging in case of errors. Two pieces of information normally found in the audit logs that the forensic examiner can use for forensic attribution are the access records and the operation records.

The access records show who connected to the database and when, while the operation records show what operations or queries were performed when and by whom. However, without authentication enabled or available there will be no access records and the operations will not have any user associated with them.

None of the surveyed NoSQL databases provided specific audit logs in the free and community versions. This means that to perform forensic attribution in those database versions, the forensic examiner will have to look at the other groups of log files.

System Logs: System or operational logs are created by the databases during the normal running of the system and can contain many different informational and error messages. How valuable these system log files are to the forensic examiner depends on their content.

The survey showed that some of the NoSQL databases include the ability to configure what messages and operations are written to the system log. This includes to a certain extent access and operation records. Thus the normal system log file can be turned into an audit log as well.

Thus if the database administrator has enabled logging and configured the system log appropriately, the forensic examiner can use them to aid forensic attribution. Unfortunately this makes the availability of system logs not something the forensic investigator can depend on when performing forensic attribution.

Storage Logs: Storage logs that are available on some of the surveyed NoSQL databases are created by their persistence mechanisms. These storage logs contain the information of all the operations that modified the data. Storage logs may or may not be archived after the information they contain has been transferred to the data files. This depends on the configuration of the database storage and available space.

The storage logs perform two functions in the NoSQL databases that use them. Firstly they speed up the write speed of the database by first writing the change operations to a small linear file before applying them to the bigger complex data files. Secondly they maintain a record of changes in case the database goes down before the change operations have been fully applied to the data files.

After a failure, the database can re-apply the operations from the storage log file to the data files to get them to the current state. In the same way the forensic examiner can use the storage logs to roll back the state of the database to an earlier point in time. This process is called reconstruction and can help identify changes that were made and information that was removed [48].

In order to save space, Cassandra uses a technique called compaction. Compaction is the process where the DBMS goes through the storage log and replaces individual operations that made changes to the same data with a single operation that has the same outcome [49]. The problem for the forensic examiner is that he no longer can see the individual operations that were performed possibly by different users.

It ultimately depends on the scenario the forensic investigator is dealing with, as to whether these storage

logs will aid forensic attribution or not. In the case where data was accessed or taken, there will be no changes to the storage log file. However, in the case where data was modified or removed there will be entries in the storage log file that could contain clues as to who was responsible.

7. CONCLUSION

The increase in the usage of NoSQL databases for data storage, analysis and retrieval in recent years has meant that more and more confidential and sensitive data is being stored in them. This has made NoSQL databases a new target for hackers and other unauthorised entities. This in turn would prompt more forensic examinations to determine which data was compromised and who was responsible. Since these NoSQL databases seem to lack adequate security features it was necessary to determine how this would impact forensic attribution.

Forensic attribution in digital systems is difficult to perform because the actions that need to be attributed occurred in the digital world. However, the initiators of these actions are located in the physical world. To be able to attribute actions to a program or process, various sources of traces are needed. These traces are then correlated to form a chain of events that can help pinpoint the initiating program or process.

A survey of four top ranked NoSQL databases was performed to determine what security measures are available, that could aid the forensic investigator to perform forensic attribution. The survey specifically looked at the areas of authentication, authorisation and logging.

Even though the surveyed NoSQL databases MongoDB and Cassandra have the same security features available as in widely used relational databases, they are not enabled and configured appropriately in default configuration mode. When performing a forensic examination, the forensic examiner is thus completely reliant on the configuration that the database administrator performed on the particular database.

Furthermore, the surveyed NoSQL databases Redis and Neo4j did not provide security features that left relevant traces. In those databases the forensic examiner is thus forced to only use traces from outside the database to help perform attribution of the actions that occurred inside the database. The lack of these traces can negatively impact the accuracy of the attribution result.

The biggest concern however, was that the surveyed NoSQL database Neo4j only provided relevant security measures in the paid for enterprise edition. This makes data security seem like an optional extra, even though many countries have laws regarding information privacy and data security. This would make Neo4j completely unsuitable for many applications in those countries. That is unless the organisations wanting to use Neo4j are prepared to pay for it.

The next step would be to determine the prevalence of free and open source editions compared to paid-for editions among deployed NoSQL databases. This would provide an idea of how many NoSQL databases would not have the necessary information at the database level to aid forensic attribution. In those cases the forensic examiner would need to rely on external systems and the network to perform forensic attribution. These same NoSQL databases would also present easier targets to hackers and other malicious actors because of their lack of available security features.

In order to make the available trace information that could aid forensic examinations more concrete, a study focussed on the detail content of the various identified log files would be required. Such a study would also have to consider multiple versions of the same NoSQL DBMS software. This is because the studied NoSQL databases are still rapidly developing and the information being logged can change between versions.

An aspect that has not been addressed in this paper is the protection provided to the log files that were identified. What methods are used to prevent tampering and what mechanisms are built into the DBMSs to detect compromised log files? Some future work would be required to answer these questions.

REFERENCES

[1] DB-Engines. (2017, Apr.) DB-Engines Ranking - Trend Popularity. [Online]. Available: https://db-engines.com/en/ranking_trend

[2] D. Feinberg, M. Adrian, N. Heudecker, A.M. Ronthal, and T. Palanca. (2015, Oct.) Magic Quadrant for Operational Database Management Systems. Gartner Inc. [Online]. Available: https://www.gartner.com/doc/3147919/magic-quadrant-operational-database-management

[3] C. Strauch. (2011) NoSQL Databases. Stuttgart Media University. [Online]. Available: http://www.christof-strauch.de/nosqldbs.pdf

[4] L. Okman, N. Gal-Oz, Y. Gonen, E. Gudes, and J. Abramov, "Security Issues in NoSQL Databases," in *Proceedings of the 10th International Conference on Trust, Security and Privacy in Computing and Communications*, Changsha, China, Nov. 16–18, 2011, pp. 541–547.

[5] S. Srinivas and A. Nair, "Security maturity in NoSQL databases," in *Proceedings of the 2015 International Conference on Advances in Computing, Communications and Informatics*, Kochi, India, Aug. 10–13, 2015, pp. 739–744.

[6] D.A. Wheeler and G.N. Larsen, "Techniques for Cyber Attack Attribution," Institute for Defense Analyses, VA, Oct. 2003.

[7] D.D. Clark and S. Landau, "Untangling Attribution," in *Proceedings of a Workshop on Deterring CyberAttacks: Informing Strategies and Developing Options for U.S. Policy*, Washington, DC, Jun. 10–11, 2010, pp. 25–40.

[8] F. Cohen, *Digital Forensic Evidence Examination*, 4th ed. Livermore, CA: Fred Cohen & Associates, 2009.

[9] M.S. Olivier, "On metadata context in Database Forensics," *Digital Investigation*, vol. 5, pp. 115–123, Mar. 2009.

[10] S. Edlich. (2017, Apr.) NOSQL Databases. [Online]. Available: http://nosql-database.org/

[11] DB-Engines. (2017, Apr.) DB-Engines Ranking. [Online]. Available: https://db-engines.com/en/ranking

[12] MongoDB Inc. (2017, Apr.) The MongoDB 3.4 Manual. [Online]. Available: https://docs.mongodb.com/manual/

[13] Apache Cassandra. (2017, Apr.) Apache Cassandra Documentation v3.2. [Online]. Available: https://cassandra.apache.org/doc/latest/

[14] Redis Labs. (2017, Apr.) Documentation. [Online]. Available: https://redis.io/documentation

[15] Neo4j. (2017, Apr.) The Neo4j Operations Manual v3.1. [Online]. Available: https://neo4j.com/docs/operations-manual/current/

[16] E. Evans. (2009, Oct.) NoSQL: What's in a name? [Online]. Available: http://blog.sym-link.com/2009/10/30/nosql_whats_in_a_name.html

[17] J. Ellis. (2009, Nov.) The NoSQL Ecosystem. [Online]. Available: https://blog.rackspace.com/nosql-ecosystem

[18] E. Eifrem. (2009, Oct.) Emil Eifrem on Twitter. [Online]. Available: https://twitter.com/emileifrem/statuses/5200345765

[19] D. Sullivan, *NoSQL for Mere Mortals*, 1st ed. Hoboken, NJ: Addison-Wesley Professional, 2014.

[20] I. Robinson, J. Webber, and E. Eifrem, *Graph Databases*, 2nd ed. Sebastopol, CA: O'Reilly Media, Inc., 2015.

[21] M.A. Olson, K. Bostic, and M. Seltzer, "Berkeley DB," in *Proceedings of the 1999 USENIX Annual Technical Conference*, Monterey, CA, Jun. 6–11, 1999.

[22] D. Harris. (2013, Aug.) 10gen embraces what it created, becomes MongoDB Inc. [Online]. Available: https://gigaom.com/2013/08/27/10gen-embraces-what-it-created-becomes-mongodb-inc/

[23] MongoDB Inc. (2017, Apr.) Introduction to MongoDB. [Online]. Available: https://docs.mongodb.com/manual/introduction/

[24] MongoDB Inc. (2017, Apr.) Our Customers — MongoDB. [Online]. Available: https://www.mongodb.com/who-uses-mongodb

[25] J. Hamilton. (2008, Jul.) Facebook Releases Cassandra as Open Source. [Online]. Available: http://perspectives.mvdirona.com/2008/07/facebook-releases-cassandra-as-open-source/

[26] DataStax. (2017, Apr.) Apache Cassandra 3.0. [Online]. Available: http://docs.datastax.com/en/cassandra/3.0/

[27] Apache Cassandra. (2017, Apr.) Apache Cassandra. [Online]. Available: https://cassandra.apache.org/

[28] S. Sanfilippo. (2010, Mar.) VMware: the new Redis home. [Online]. Available: http://oldblog.antirez.com/post/vmware-the-new-redis-home.html

[29] Redis Labs. (2017, Apr.) Introduction to Redis. [Online]. Available: https://redis.io/topics/introduction

[30] Redis Labs. (2017, Apr.) Who's using Redis? [Online]. Available: https://redis.io/topics/whos-using-redis

[31] Neo4j. (2017, Apr.) Chapter 1. Introduction. [Online]. Available: https://neo4j.com/docs/operations-manual/current/introduction/

[32] Neo4j. (2017, Apr.) Neo4j Cypher Refcard 3.1. [Online]. Available: https://neo4j.com/docs/cypher-refcard/current/

[33] Neo4j. (2017, Apr.) Neo4j Customers. [Online]. Available: https://neo4j.com/customers/

[34] H.H. Kelley, "The processes of causal attribution," *American Psychologist*, vol. 28, no. 2, pp. 107–128, Feb. 1973.

[35] M. Wallenius, K. Mayer, and I. Ray, "Nuclear forensic investigations: Two case studies," *Forensic Science International*, vol. 156, no. 1, pp. 55–62, Jan. 2006.

[36] P. Juola, "Authorship Attribution," *Foundations and Trends in Information Retrieval*, vol. 1, no. 3, pp. 233–334, Mar. 2008.

[37] W.E. Boebert, "A Survey of Challenges in Attribution," in *Proceedings of a Workshop on Deterring CyberAttacks: Informing Strategies and Developing Options for U.S. Policy*, Washington, DC, Jun. 10–11, 2010, pp. 41–52.

[38] ISO, "Information technology – Security techniques – Incident investigation principles and processes," International Organization for Standardization, Geneva, Switzerland, ISO/IEC 27043:2015, Mar. 2015.

[39] F.B. Cohen, "Attribution of messages to sources in digital forensics cases," in *Proceedings of the 43rd Hawaii International Conference on System Sciences*, Honolulu, HI, Jan. 5–8, 2010, pp. 4459–4468.

[40] J.A. Shamsi, S. Zeadally, F. Sheikh, and A. Flowers, "Attribution in cyberspace: techniques and legal implications," *Security and Communication Networks*, vol. 9, no. 15, pp. 2886–2900, Oct. 2016.

[41] T. Rid and P. McBurney, "Cyber-Weapons," *The RUSI Journal*, vol. 157, no. 1, pp. 6–13, Feb. 2012.

[42] W.K. Hauger and M.S. Olivier, "Determining trigger involvement during Forensic Attribution in Databases," in *Advances in Digital Forensics XI*, G. Peterson and S. Shenoi, Eds. Heidelberg, Germany: Springer, 2015.

[43] ISO, "Information technology – Database languages – SQL – Part 2: Foundation (SQL/Foundation)," International Organization for Standardization, Geneva, Switzerland, ISO/IEC 9075-2:2012, Oct. 2012.

[44] Neo4j. (2017, Apr.) Compare Neo4j Editions. [Online]. Available: https://neo4j.com/editions/

[45] C. Newman, A. Menon-Sen, A. Melnikov, and N. Williams, "Salted Challenge Response Authentication Mechanism (SCRAM) SASL and GSS-API Mechanisms," Internet Requests for Comments, RFC Editor, RFC 5802, July 2010. [Online]. Available: http://www.rfc-editor.org/rfc/rfc5802.txt

[46] Oracle. (2017, Apr.) Java Logging Package. [Online]. Available: https://docs.oracle.com/javase/6/docs/api/java/util/logging/package-summary.html

[47] K. Seguin, *The Little Redis Book*. Self-Published, 2012.

[48] O.M. Adedayo and M.S. Olivier, "Ideal log setting for database forensics reconstruction," *Digital Investigation*, vol. 12, pp. 27–40, Mar. 2015.

[49] J. Ellis. (2011, Oct.) Leveled Compaction in Apache Cassandra. [Online]. Available: http://www.datastax.com/dev/blog/leveled-compaction-in-apache-cassandra