Forensic Attribution Challenges During Forensic Examinations Of Databases

by

Werner Karl Hauger

Submitted in fulfilment of the requirements for the degree
Master of Science (Computer Science)
in the Faculty of Engineering, Built Environment and Information Technology
University of Pretoria, Pretoria

September 2018

Publication data:

Werner Karl Hauger. Forensic Attribution Challenges During Forensic Examinations Of Databases. Master's dissertation, University of Pretoria, Department of Computer Science, Pretoria, South Africa, September 2018.

Electronic, hyperlinked versions of this dissertation are available online, as Adobe PDF files, at:

https://repository.up.ac.za/

Forensic Attribution Challenges During Forensic Examinations Of Databases

by

Werner Karl Hauger
E-mail: whauger@gmail.com

Abstract

An aspect of database forensics that has not yet received much attention in the academic research community is the attribution of actions performed in a database. When forensic attribution is performed for actions executed in computer systems, it is necessary to avoid incorrectly attributing actions to processes or actors. This is because the outcome of forensic attribution may be used to determine civil or criminal liability. Therefore, correctness is extremely important when attributing actions in computer systems, also when performing forensic attribution in databases. Any circumstances that can compromise the correctness of the attribution results need to be identified and addressed.

This dissertation explores possible challenges when performing forensic attribution in databases. What can prevent the correct attribution of actions performed in a database? The first identified challenge is the database trigger, which has not yet been studied in the context of forensic examinations. Therefore, the dissertation investigates the impact of database triggers on forensic examinations by examining two sub questions. Firstly, could triggers due to their nature, combined with the way databases are forensically acquired and analysed, lead to the contamination of the data that is being analysed? Secondly, can the current attribution process correctly identify which party is responsible for which changes in a database where triggers are used to create and maintain data? The second identified challenge is the lack of access and audit information in NoSQL databases. The dissertation thus investigates how the availability of access control and logging features in databases impacts forensic attribution.

The database triggers, as defined in the SQL standard, are studied together with a number of database trigger implementations. This is done in order to establish, which aspects of a database trigger may have an impact on digital forensic acquisition, analysis and interpretation. Forensic examinations of relational and NoSQL databases are evaluated to determine what challenges the presence of database triggers pose. A number of NoSQL databases are then studied to determine the availability of access control and logging features. This is done because these features leave valuable traces for the forensic attribution process. An algorithm is devised, which provides a simple test to determine if database triggers played any part in the generation or manipulation of data in a specific database object. If the test result is positive, the actions performed by the implicated triggers will have to be considered in a forensic examination.

This dissertation identified a group of database triggers, classified as non-data triggers, which have the potential to contaminate the data in popular relational databases by inconspicuous operations, such as connection or shutdown. It also established that database triggers can influence the normal flow of data operations. This means what the original operation intended to do, and what actually happened, are not necessarily the same. Therefore, the attribution of these operations becomes problematic and incorrect deductions can be made. Accordingly, forensic processes need to be extended to include the handling and analysis of all database triggers. This enables safer acquisition and analysis of databases and more accurate attribution of actions performed in databases. This dissertation also established that popular NoSQL databases either lack sufficient access control and logging capabilities or do not enable them by default to support attribution to the same level as in relational databases.

Keywords: Digital Forensics, Database Forensics, Forensic Attribution, Database Triggers, Relational Databases, NoSQL Databases.

Supervisor : Prof. Martin S. Olivier
Department : Department of Computer Science
Degree : Master of Science

"We choose to ... do the other things, not because they are easy, but because they are hard, because that goal will serve to organize and measure the best of our energies and skills, because that challenge is one that we are willing to accept, one we are unwilling to postpone, and one which we intend to win."

John F. Kennedy (1963)

"Quod scripsi, scripsi (What I have written, I have written)"

Pontius Pilate (AD 36)

Acknowledgements

I wish to express my sincere thanks and gratitude to:

- My parents who supported my decision to return to academia and continue with my studies. You supported and encouraged me throughout this endeavour;
- Professor Martin S. Olivier for reactivating my academic mindset and his help, guidance and support over all the years;
- All the members of the ICSA research group with whom I could exchange ideas and discuss my research (Victor, Stacy, Richard, Avinash).

This dissertation is dedicated to my late father who unfortunately could not see its conclusion.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Databases have, in some form or another, been used in computer systems since the early 1960s. They were first used by governmental organisations, big corporations and financial institutions that also had the first computer systems. With the advent of general-purpose database systems based on the relational model, databases became more common in the 1970s. When the desktop computer emerged in the 1980s, databases moved into the offices and homes of users (Elmasri & Navathe, 1994, p. 21).

With the advent of mobile devices in the 1990s, databases slowly moved into the pockets of users. A great deal of mobile applications (apps) now persist and manipulate their data utilising compact and modest databases such as SQLite. Thus, databases can no longer be considered niche products. In fact, databases are probably now used everywhere where there is a computer system. In this age of Big Data, everyone with a computer system appears to want to persist, query and manipulate data.

The normal day to day users of these modern systems and applications have become completely unaware of the databases that are used inside them (Silberschatz, Korth, & Sudarshan, 2011, p. 3). That is because the databases reside and operate in the background and are never seen by the user of the system or application. Rather, the users interact with these systems and applications through front-ends that in turn retrieve, manipulate and store the data in the databases.

However, groups that have long ago recognised the value of databases include criminals and malicious agents. They have understood that databases can be accessed and

manipulated to perform malicious and criminal activities (Casey, 2009, p. 2). These groups constantly find new ways to compromise and abuse databases in computer systems and applications. Security practitioners have followed these malicious and criminal activities and learned how to detect such activities on databases and how to protect databases against such attacks. They have published books and training guides on how to secure specific databases and how to perform forensic examinations on compromised databases (Fowler, 2009; Litchfield, 2007).

Other groups, such as auditors and law enforcement agencies, have realised that computer systems and their databases can be valuable sources of information to help them with their tasks. In many cases, the systems and applications track all the activities of their users and store them in the database (Casey, 2009). Besides the activities themselves, the database can also contain useful information such as timestamps and Global Positioning System (GPS) locations. This type of information can be used by law enforcement agencies in various ways. It can provide digital evidence to implicate specific persons of unlawful activity. It can also provide digital evidence that confirms an alibi and thus exonerates specific persons. Auditors can use this information to confirm that a computer system is being used as declared and that the correct duly authorised persons operate and use the computer system.

The forensic examinations of databases fall under the broader field of digital forensics. There has been a steady increase in scientific research in various other areas of digital forensics in recent years (see chapter 4). However, the scientific community has not kept pace with the developments in the database forensics area. Such research is needed to make sure that these practices, which were developed and are performed, are scientific and correct.

1.1 Motivation

One particular area in forensics where this correctness is very important is the process of attribution. On the highest level, attribution deals with the question of “who did it”. Thus, attribution attempts to find the responsible actor (individual or group) for a particular action or event. Therefore, it would follow that always identifying the correct

actor is important. However, this actually depends on the reason why attribution is being performed. If the goal is to act as a deterrence to prevent similar actions in the future, the correct parties need to be identified and held accountable. Innocent parties should not be implicated and punished. On the other hand, if the goal is to simply mount a better defence against the actions that are being attributed, it might be sufficient to identify possible groups and individuals.

Besides disparate starting goals, there are other reasons why all attribution being performed is not of the same specificity. In their paper, Rid and Buchanan (2015) contend that attribution is as much art as it is science. They describe attribution as being a whole process that uses many technical and non-technical tools and skills that are applied at various levels by various experts. Putting all this together provides attribution to a certain degree, which is neither black-or-white, but rather a certain shade. In addition, Rid and Buchanan argue that the accuracy of attribution depends on the political will behind the attribution, because this determines the time and resources available.

In his work, Boebert (2010) breaks the attribution question down into two separate attribution problems: technical attribution and human attribution. Technical attribution takes place in the digital world and analyses various traces to find the responsible machine (the what) from where the questionable action or event was launched. Human attribution then takes place in the physical world and takes the outcome of the technical attribution and combines it with other information from the physical world to identify the actor (the who) responsible.

Performing attribution in the physical world should be considered outside the scope of digital forensics. This is because in the physical world attribution might require the use of other expertise and non-technical means. This element is problematic for two reasons. Firstly, the required expertise normally falls outside the technical speciality of the digital forensic examiner. Secondly, some of the means may not be scientific in nature but are rather based on intuition and "gut feel".

In the digital world, however, things are simpler. All high-level events and actions follow pre-programmed processes and algorithms. They in turn are executed by processors based on finite state machines. This means that there are predictable transitions and a fixed number of known outcomes. Processes and algorithms executed by such

finite state machine based processors will thus always, barring outside interference such as electromagnetic interference (EMI), produce the same traces and outcomes given the same input.

This means that the traces left by these processes and algorithms can be used to make scientifically sound inferences and conclusions about the events that generated them. Because the traces are left by finite state machines, the inferences can be scientifically proven to be correct. These inferences and conclusions that are made might not necessarily help in the greater attribution process of identifying a root cause (the what). However, standing on their own, these inferences and conclusions should always be correct.

The correctness of any attribution technique or tool that is being used in the digital world is extremely important. Otherwise, failure to ensure absolute correctness or correctness with a mathematically insignificant small error can lead to wrong conclusions being made. These wrong conclusions can then lead to actions, consequences and outcomes in the physical world that potentially cannot be undone. Therefore, any unintended effects, reactions or outcomes that occur when the attribution techniques and tools are used, need to be known and managed properly.

These unintended effects, reactions or outcomes can be seen akin to the side-effects that occur with the administration of drugs in the medical field. A drug has been created and designed to have a specific effect on the human body. However, many drugs do not only have the one desired effect, but may also have other unintended and unwanted effects.

For example, aspirin was developed to treat the symptoms of pain and fever. However, aspirin also suppresses platelet functioning leading to blood-thinning properties (Burch, Stanford, & Majerus, 1978). When administered to a person without any disorders, the aspirin will have the intended effect. However, when administered to a person with a bleeding disorder, the aspirin can lead to uncontrollable bleeding and possibly death. In this case, the side-effect of aspirin can completely overshadow the intended positive effect and lead to a very negative outcome.

The same can happen with performing attribution. The attribution technique or tool A may be utilised to achieve an intended outcome or effect E_1. Nevertheless, there may

be additional effects $E_2..E_n$ in play, whether initiated by the attribution technique or tool, or being part of the forensic artefact under examination. These additional effects $E_2..E_n$ may cause doubts about the intended effect E_1 or may even overshadow the intended effect E_1 completely leading to wrong conclusions.

In the area of database forensics, one mechanism that has the potential to create such side-effects is the database trigger. A trigger is an automated action that occurs in reaction to an event taking place in the database. A simple application would be a trigger that is placed on a particular column in a table that is stored in a relational database. As an example, the trigger is configured to operate as follows: every time any value in the column is updated to a specific value, the trigger will perform an update on a related value in another table.

Should attribution now be performed on the change of the related value, the correct answer is no longer straightforward. Is the root-cause of the changed value the changing of the original value or the presence of the trigger? The correct answer depends on the context of the examination and the forensic examiner needs to be made aware of all possible answers. Otherwise, a scenario where a database trigger might influence the correctness of the attribution being performed, is plausible. Incorrect attribution scenarios due to side-effects are nothing new. A well-known example of an attribution technique that is used in the physical world, and which suffered from inaccuracies due to side-effects, is DNA profiling.

Various DNA typing techniques have been developed and scientifically proven for diverse types of DNA samples. This means that these tests can be independently repeated with the same samples and provide the same results every time. The correctness is guaranteed to a very high mathematical probability if the sample meets certain physical and origin requirements (National Research Council, 1996). In the homogeneous populations where the DNA tests were originally developed, the error rate was so low that no accidental match was realistic.

However, as the usage of DNA profiling increased and spread all around the world, a number of side-effects were discovered. The allele distribution frequency of the original loci that were chosen as sufficient differentiators between individuals, was too different in other population groups of the world (National Research Council, 1996). This increased

the probability of an accidental match between two unrelated DNA samples.

Thus, the selection of additional sets of loci to support different population groups became necessary. DNA databases also needed to be created for every significant population group. These databases are used to determine the allele frequencies for all tested loci data. During DNA profiling, these databases are consulted to determine whether the usage of a specific locus is appropriate for a specific population group (National Research Council, 1996).

Furthermore, in heterogeneous populations the use of just one set of loci was ineffective since it would only have the necessary frequency spread for one population group. Thus, single loci sets had to be combined to create extended sets with more alleles that could be used to differentiate between individuals in the heterogeneous populations (Butler & Hill, 2012). The product rule then ensured that the combined probability of an accidental match remained very low (National Research Council, 1992).

The envelope was also being pushed with regards to the quality of DNA samples used for testing. The promise of leads or answers to questions in those cases led to DNA tests being performed on old and degraded samples, contaminated samples and damaged samples. As was discovered, those samples could produce DNA profiles with faint or completely missing alleles or to alleles of incorrect length (Thompson, Mueller, & Krane, 2012). One special type of contamination was the presence of DNA from multiple individuals in a sample. Such samples are called mixtures and the correct separation of alleles becomes problematic (Lubaale, 2015). Just a single incorrectly allocated allele would produce two different DNA profiles in a mixture of two DNA samples. Depending on the distribution of the incorrect alleles in the population group, the incorrect DNA profile could actually match the DNA profile of a completely unrelated individual.

Another area where DNA testing was also introduced was for samples collected from the surface of objects. Tiny amounts of skin cells can be left on objects after they have been touched or handled by an individual. These samples are called touch DNA. It was quickly discovered that using touch DNA can be problematic because it can be transferred. Originally it was thought that secondary touch DNA on an object could always be identified as such and thus be excluded. However, recent research has shown that this is not always the case (Cale, Earll, Latham, & Bush, 2016). In specific circumstances, it

is possible for an object to have no touch DNA of the primary individual who actually touched the object, but rather the DNA of a secondary individual who has had contact with the primary individual, but never with the object itself.

As the above example shows, even a scientifically proven technique such as DNA profiling can produce incorrect results if previously unknown side-effects occur during specific use cases. Forensic techniques and tools used in the digital world will not be immune to incorrect results due to unknown side-effects. Consequently, research that looks at new and unknown side-effects is needed to ensure better results of the forensic tools and techniques that are being used.

1.2 Objective

Current database forensic research focusses mainly on two areas: The first research area deals with how to perform forensic examinations of databases. This research highlights commonalities and differences to other forensic examination fields such as file system forensics and network forensics. This research also proposes new examination models and processes to help with the task of examining a database in a forensically sound manner.

The second research area deals with identifying what changes were made in database to particular objects, especially when those changes were malicious and unauthorised. This research provides new techniques and algorithms that can be used to find changes, such as modifications and removals, which are no longer directly available. Many of these techniques are developed to also handle cases where the perpetrators actively tried to hide and obfuscate their actions. However, much of this research is particular to specific Database Management System (DBMS) platforms and implementations.

Both research areas appear to work on the premise that the databases that need to be examined are either corrupted or compromised. The corruption might be due to the deletion of parts or the whole of the database. This could have been done on purpose in order to destroy potential evidence or by accident. Another cause for database corruption might be physical damage to the medium or device that the database is stored on. In the case of a compromise, it is assumed that the perpetrator might have placed anti-forensic

tools to hide his presence and actions (Casey, 2009, p. 2). Alternatively, the perpetrator might have modified and deleted traces to conceal his presence and actions. Therefore, when a forensic examination is performed, all data is extracted using first principles and analysed with clean tools.

The current research appears to imply that a forensic examination is only performed when the database was breached and there is some foul play involved. However, data breaches and suspected foul play are not the only reason why the forensic examination of a database may be required. Databases can also be examined purely for the purpose of the information they contain and the deductions that can be made from this information. For example, as part of an audit of an information system to confirm the required segregation of duties, the audit trail stored in a database can be examined. Database permissions and logs might also be used to confirm that no violations of segregation are possible or have occurred during the audit period.

Another example where the forensic examination of databases may be required is during an investigation into the cause of an information system failure. The investigation may be internal to identify responsibility, or it may be an external investigation to determine criminal or civil liability. In both examples, the database will be in a working condition and no foul play is indicated or initially suspected. In these cases, the information contained in the database is taken at face value. The database and its tools are used to extract and analyse the data to retrieve the required information.

The information gained from the database can be correlated with other gathered information (Casey, 2009, p. 27). The other information may originate from the database as well, from other components of the digital system or even from the physical world. If the other information corroborates the knowledge gained from the database, then no further action is required. Should the other information, however, contradict the database information, then there is a possibility of database tampering and a deeper analysis is required. It is, however, still not a given that the database has been compromised because the other information source might just as well have been the target of a compromise. The information source, which is more likely to be incorrect in the particular circumstances, should be the first target of a more in-depth examination.

Very limited database forensic research is conducted that specifically focusses on iden-

tifying who is responsible for accessing and/or changing information stored in databases (see chapter 4). Given the previous two examples, it would follow that this type of database examination is often required, especially given the increased usage of information systems in all areas of the modern society. This described area of research can be categorised as forensic attribution because it attempts to find the responsible process (or what) in the digital world and the responsible actor (or who) in the physical world.

This lack of research in the area of attribution in databases could be due to what Pollitt (2013) and Rid and Buchanan (2015) said about forensic attribution being a combination of art and science. The portion of forensic attribution that is considered art would not be formalised and written down. That means it cannot be scientifically evaluated and tested for correctness. The science portion of forensic attribution would refer to the attribution techniques and methods that have been scientifically developed and tested for correctness. These attribution techniques and methods are the topics of papers that have been published for public use and scientific scrutiny.

Research into the side-effects of triggers during forensic examinations of databases has not yet been done. Thus far, only Khanuja and Adane (2012) have recognised the potential influence of triggers on forensic examinations of a database. They have, however, not analysed this further in their research.

The objective of this dissertation is to make contributions to both those areas of database forensic research. Knowledge will be added to the forensic examination processes as well as the activity of attribution. Specifically, this dissertation attempts to help ensure that any deductions and conclusions made from database traces found in memory, logs and database system tables are correct. This includes taking into account any hidden side-effects that can occur in a database, whether it be due to the normal operation of the database or due to malicious intentions and actions.

1.3 Problem Statement

This research explores possible unexpected challenges to the process of forensic attribution. The challenges of particular interest to this dissertation are those that can prevent correct attribution in the digital world of the database. The specific research question

that needs to be answered is the following: What circumstances can prevent the correct attribution of actions performed in a database?

This question is examined in the context of forensic examinations of the current generation of relational and NoSQL databases. The scope of this research is restricted to relational and NoSQL databases because these two are the most prevalent database models used in computer systems (DB-Engines, 2018).

One candidate that could pose a challenge to forensic attribution is the mechanism that was already introduced in section 1.1: the database trigger. Given that no research on database triggers in relation to forensic examinations has been done yet, it might be worthwhile to expand the research question with regards to database triggers as follows: Do database triggers interfere with forensic examinations?

This question needs to be answered for at least two separate phases of the forensic examination procedure. The first phase deals with the process of forensic acquisition and forensic analysis of a database. The question that needs to be answered here is the following: Do the current forensic acquisition and analysis processes make provision for the presence of triggers during a forensic examination of a database?

The second phase deals with the process of interpreting the data from the acquired database. Specific techniques or processes that are used in this step include reconstruction and attribution. The process of interest that is possibly vulnerable is attribution. The question that needs to be answered here is the following: Can attribution be accurately performed when dealing with actions that include those performed by triggers?

1.4 Approach

In order to answer the expanded research question, it is first necessary to understand exactly what database triggers are and what actions they can accomplish. To determine this, a survey of current available database triggers is performed. Firstly, the SQL standard is examined to identify all possible triggers and their operation, as defined. Then, a number of relational database implementations are analysed to confirm the presence of the triggers as defined in the standard. The same relational database implementations are also analysed to identify any new triggers or triggers that deviate from the standard.

Once the available database triggers are known and understood, focus is placed on general database forensics. The digital forensic processes that are followed during a forensic examination of a relational database are evaluated. This is achieved by performing a case study. The steps that are performed in a published forensic database examination are scrutinised to determine if the presence of any of the known triggers could influence or alter the outcomes as documented. Should any possible alterations be identified, the implications of these changes are discussed and counter-measures contemplated.

Then the focus is narrowed down to the forensic interpretation process. Specifically, the activity of forensic attribution is examined. Attribution techniques that are used in database examinations are scrutinised to determine if the presence of any of the known triggers could influence the root-cause as determined by the technique. Should the determined initiators be influenced by triggers, it becomes necessary to confirm under which circumstances the presence of triggers can be problematic.

Once the exact circumstances are known, an algorithm can be proposed to safely test for the presence of database triggers. As a minimum, this will alert the forensic examiner to be careful when applying attribution techniques to determine the initiators such as processes and user profiles. Should the database being examined contain many triggers, the algorithm can be extended to be applied to the specific database objects that the forensic examiner is using to perform the attribution. The algorithm will then identify specific triggers that the forensic examiner will need to examine for possible influence on the attribution results.

1.5 Dissertation Outline

The remaining chapters of this dissertation are structured as follows:

- **Chapter 2** provides an overview of the first of two core elements that form the foundation of this dissertation: database systems. The historical development of the various database models and types that exist today is presented. Special attention is given to relational and NoSQL databases, which constitute the central part of this dissertation.

- **Chapter 3** continues with an overview of the second of the two core elements that

form the foundation of this dissertation: digital forensics. The concept of digital forensics is described with specific emphasis on database forensics. Attribution, one of the processes that can be used when performing forensic examinations, is discussed. Various forensic attribution types and techniques are reviewed.

- **Chapter 4** explores other research in the field of database forensics. Due to the limited amount of research available, the chapter performs a survey of all the major database forensic research performed since the first scholarly research on database forensics was peer-reviewed. Then, the chapter also probes the possible reasons for the lack of research in this field of database forensics.

- **Chapter 5** takes an in-depth look at database triggers and their workings. The database trigger implementations of several relational database management systems are analysed and evaluated for possible interference in forensic examinations.

- **Chapter 6** studies the forensic examination of relational databases. Particular emphasis is placed on the possible implications that the presence of database triggers can have on the forensic examination. This includes possible interference of database triggers with the interpretative process of attribution.

- **Chapter 7** proposes an algorithm that can be used to identify potentially interfering database triggers. Two variations of the algorithm are presented and their implementability is evaluated.

- **Chapter 8** discusses the implementation of the algorithm proposed in chapter 7. A prototype implementation of one variation of the algorithms is presented and the challenges encountered during implementation and usage are analysed.

- **Chapter 9** moves the focus to the forensic examination of NoSQL databases. The impact of triggers in this model of database is briefly discussed. The challenge that the absence of certain security measures in these databases presents to forensic attribution is then discussed in detail. Firstly, a survey is performed to ascertain the default availability of these security measures in popular NoSQL databases. Then the impact that the lack of security measures has on the process of forensic attribution is established.

- **Chapter 10** concludes this dissertation by summarising the major findings. Future research is also contemplated based on the outcomes presented.

The following appendices are part of this dissertation:

- **Appendix A** provides a list of the important acronyms used in this work, as well as their associated definitions.
- **Appendix B** lists all the publications that were derived from this work.

Chapter 2

Database Systems Overview

The aim of this and the following chapter is to provide an overview of some of the core systems and aspects that underpin this dissertation. The key element that is discussed in this chapter is the database system. The chapter first looks at the research and development history of database systems. The discussion of two specific database models and a database concept then follows.

Section 2.1 provides a historical background of database development by discussing the major data models that were developed over the past five decades. Also, the main actors involved in this research and development and the major products that came out of this development are presented. In section 2.2, specific attention is given to relational and NoSQL databases, which are the focus of this research. In section 2.3, the concept of a database trigger is explored in more detail. Section 2.4 concludes this chapter.

2.1 Database History

A database can be defined as a collection of related data (Elmasri & Navathe, 1994). Furthermore, a DBMS is a set of programs that allows users to create, access and maintain this database. Together the database and the software form a database system.

Database systems arose in response to earlier methods of computerised management of data such as files. File processing systems have quite a number of problems and disadvantages. They include data redundancy and inconsistency, data access difficulties, data

isolation, integrity and atomicity problems, and concurrency and security problems (Silberschatz et al., 2011). This prompted the development of alternative data management systems.

A chronological discussion of the major database development eras, and the systems for data management developed during them, now follows.

2.1.1 Network Era

The first attempt at a data management system to overcome the limitations of files was the Integrated Data Store (IDS) developed by Charles Bachman at General Electric in 1963. The IDS system formed the basis for the network data model developed by the CODASYL Database Task Group (DBTG) in 1969 (Haigh, 2011).

In the network model, related data is represented as records. A format called a record type is used to describe the structure of a record (Elmasri & Navathe, 1994). Each record type has a name and consists of a number of data items or attributes. Each data item also has a name and a format or data type.

Different record types can be related to each other by set types (Elmasri & Navathe, 1994). A set type describes a 1:N relationship between two record types. Diagrammatically a set type is represented by an arrow. Each set type has a name and defines the owner record type and the member record type. Set instances consist of a single owner record of the owner record type and one or more member records of the member record type.

Due to the 1:N relationship between two record types in a set type, a member record can only appear in one set instance. A 1:1 relationship can be represented by only having a single member record in a set instance. However, a M:N relationship cannot be represented by a single set type between two record types. A third record type, called a linking or dummy record type, is required (Elmasri & Navathe, 1994). One set type is created between the first record type and the linking record type and another between the second record type and the linking record type. A record of the linking record type will then be a member record of both the first and the second set type.

2.1.2 Hierarchical Era

In 1966, IBM launched their first attempt at a data management system called the Information Management System (IMS). Chief architect Vern Watts led the development of a system where data was organised in multiple levels. This concept formed the basis for the hierarchical data model.

In the hierarchical model, related data is also stored using record types that consist of data items. Relationships between two record types are 1:N in the hierarchical model. The record type on the 1-side is called a parent record type and the record type on the N-side a child record type (Elmasri & Navathe, 1994).

This relationship is called a parent-child relationship (PCR) type (Elmasri & Navathe, 1994). An instance of a PCR type would contain one parent record and zero or more child records. A number of record types and PCR types can be associated together in a hierarchical schema or hierarchy. Hierarchical schemas can be visually displayed in a hierarchical diagram with rectangular boxes and lines connecting them.

Hierarchical schemas also have a number of properties and restrictions. The one record type that does not participate as a child record type in any PCR type is called the root. On the other end side, a record type that does not participate as a parent in any PCR type is called a leaf. While a record type can act as a parent record type in any number of PCR types, it can act as a child record type in only one PCR type.

The previous restrictions thus allow both 1:1 and 1:N relationships. However, M:N relationships cannot be represented. Only by deviating from the strict hierarchical model by allowing either the duplication of child record instances or the creation of virtual PCR types between two different hierarchical schemas, can M:N relationships be represented.

2.1.3 Relational Era

In 1970, IBM research fellow Edgar F. Codd (1970) published his relational model for data organisation. This model included data manipulation operations based on relational algebra. Two major research projects then set out to implement and test this new relational model.

One research team worked at IBM's San Jose Research Laboratory. The database

system they constructed was called System R (Chamberlin et al., 1981). In one of the implementation phases, a non-procedural query language called SEQUEL was also developed that allowed the usage of Codd's data operations. SEQUEL was later renamed to SQL due to an existing trademark.

IBM kept the source code for System R closed and used it to build two commercial relational DBMSs (RDBMSs). In the early eighties, IBM first released SQL/DS and then DB2 for various mainframe operating systems (Atzeni, Ceri, Paraboschi, & Torlone, 1999). However, IBM did not manage to release the first commercial RDBMS. In 1977, Larry Ellison had formed his own company, which released Oracle for commercial use in 1979 (Lazenby & Lindquist, 2007).

At around the same time as IBM's team, another research team worked on a relational database at the University of California, Berkeley (UCB). The team around Michael Stonebraker developed their own implementation of the relational model called INGRES (Stonebraker, Held, Wong, & Kreps, 1976). As a consequence they also developed their own query language called QUEL. Due to both languages being based on Codd's work, they had many similarities.

Unlike IBM, Berkeley made the source code to INGRES available for a fee. This led to the development of a number of new commercial RDBMSs in the early eighties, often by members of the original research project. These new RDBMSs included Sybase, Informix and NonStop SQL (Epstein, 2013; Mendelsohn, 2013; Sippl, 2013). INGRES was also commercialised by Stonebraker. In 1992, Sybase licensed their technology to Microsoft, who re-branded it as Microsoft SQL Server.

Structured Query Language (SQL) was adopted as a standard by the American National Standards Institute (ANSI) in 1986 and the International Organization for Standardization (ISO) in 1987. The standard was called SQL-86 and is also referred to as SQL1 (Elmasri & Navathe, 1994). The standardisation of the SQL version query language meant that the QUEL version, and RDBMSs using it, lost traction. It forced these RDBMSs to switch over to SQL and eventually even INGRES had to add SQL to their database.

The next major revision of the standard was SQL-92, which is also referred to as SQL2. It was a much expanded standard that added many features that had already

been implemented by various RDBMS manufacturers. The next iteration of the ISO/IEC 9075 standard came in 1999 and was called SQL:1999 or SQL3 (Silberschatz et al., 2011).

2.1.4 Object Oriented Era

In the 1980s, databases started to appear in many application domains. However, there were some more complex application domains where existing database systems had certain shortcomings. These included engineering design, image and multimedia processing and geographic information systems (Elmasri & Navathe, 1994). Thus, object-oriented databases were proposed to meet those needs. They supported object-oriented principles, which allowed database developers to specify their own complex object types and the operations that they support.

A manifesto was released by Atkinson et al. (1989) that defined what constituted an object-oriented database system (OODBMS). This included a new data language and a new query language, which were both based on object-orientation. Essentially they took a revolutionary approach by extending the DBMSs based on the characteristics of object-oriented programming languages (Atzeni et al., 1999).

2.1.5 Object Relational Era

A group of researchers around Michael Stonebraker had divergent ideas on how to solve the shortcomings of relational databases in the more complex application domains. They released their own reply manifesto that defined a third generation of database systems (Stonebraker et al., 1990). The group agreed on the usage of object orientation, but differed on how to achieve this.

This third generation of database system assumed an evolutionary approach, by integrating the object concept into the relational model (Atzeni et al., 1999). This included retaining SQL as the data and query language, but extending it with declarative elements. Some of these extensions eventually became part of the SQL3 standard. These third generation database systems became known as object-relational database systems (ORDBMSs).

Stonebraker implemented these ideas in a new research project called POSTGRES.

This ORDBMS used a query language call POSTQUEL which was based on the QUEL language of INGRES. In 1994, this query language was replaced with SQL, and subsequently the name of the DBMS was updated to PostgreSQL to reflect the new SQL support.

2.1.6 NoSQL Era

The 2000s saw a huge explosion in the usage of databases to support the growth of the internet. As the internet grew, so did transaction and data volumes that pushed existing RDBMSs and ORDBMSs to the limit. This created a new movement that felt that the rigid relational model and the restrictions of SQL limited the future growth of databases. New web companies such as Amazon, Facebook, Google and Yahoo started to develop their own alternative database systems, particularly around column stores (Silberschatz et al., 2011). These original alternative databases and their successors are now grouped together under the moniker of NoSQL databases.

2.2 Relevant Database Models

The relational database is a very established and well known entity. It has been studied and used for decades, which has led to an extensive SQL standard. Consequently, only a brief discussion is needed. The NoSQL database, on the other hand, is a new entity that is still developing and changing. Therefore, a greater effort is required to provide structure to the great variations of databases that are placed under the NoSQL umbrella. Accordingly, a more in-depth discussion is provided.

2.2.1 Relational Databases

A relational database is based on the relational model and uses a collection of tables to represent both data and the relationships among those data (Silberschatz et al., 2011). A table consists out of multiple columns and each column has a unique name. The data that the table contains, fills up multiple rows.

The relational model was first proposed by Codd (1970). He represented data as

relations (tables) and tuples (rows). The use of relations allowed him to use relational algebra to manipulate the data. This included the usual set operations such as UNION, INTERSECTION, SET DIFFERENCE and CARTESIAN PRODUCT (Codd, 1979). He also defined new operations specifically for the relational model in databases. They included THETA-SELECT, PROJECT, THETA-JOIN and DIVIDE (Codd, 1979).

A relational database also includes a Data Manipulation Language (DML) and a Data Definition Language (DDL). The database schema is specified by a set of definitions, which are expressed by a special language called a data definition language. The DDL is also used to specify additional properties of the data.

A data manipulation language is a language that enables users to access or manipulate data as organised by the appropriate data model. The four types of data access are: retrieval of information stored in the database, insertion of new information into the database, deletion of existing information from the database and modification of information stored in the database (Silberschatz et al., 2011).

2.2.2 NoSQL Databases

The NoSQL movement was the development of new types of databases that were not relational and did not use the SQL as data access language. These new database types were being created to address new demands in data forms and size that 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 (Evans, 2009). Evans 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 (Eifrem,

2009; Ellis, 2009), as 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 the current information society.

In recent years, NoSQL databases have gained popularity both with developers who build new systems, and existing organisations who try to optimise and improve their businesses (DB-Engines, 2017b). Both parties are trying to adapt their information systems to the current data demands.

Certain NoSQL databases have even moved up from being niche products to leaders in Gartners Magic Quadrant for Operational Database Management Systems (Feinberg, Adrian, Heudecker, Ronthal, & Palanca, 2015). 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 the current NoSQL databases were started by big web companies, such as Google, Amazon and Facebook, to help them build and support their businesses (Strauch, 2011). 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 (Strauch, 2011). This has led 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.

2.2.3 NoSQL Database Types

As of now, a number of distinct types of NoSQL databases have established themselves. 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 (Sullivan, 2014). 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 (Sullivan, 2014).

Document stores are characterised 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 (Sullivan, 2014). This then allows the records to be processed directly in applications. Individual documents are stored and retrieved by means of a key. Furthermore, document stores rely on indexes to facilitate access to documents based on their attributes (Robinson, Webber, & Eifrem, 2015).

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 (Sullivan, 2014).

Any number of columns can be combined into a super column, which gives a name to a sorted collection 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 (Robinson et al., 2015).

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 (Sullivan, 2014).

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 need to be unique within a namespace. A namespace could correspond to an entire database, which means all keys in the database need to 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 (Sullivan, 2014).

These plain 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 the 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 (Olson, Bostic, & Seltzer, 1999).

Graph Stores

Graph stores, also known as graph databases, are DBMSs with Create, Read, Update, and Delete (CRUD) methods that manipulate 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 (Sullivan, 2014).

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 (Robinson et al., 2015). 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 distinct graph data models commonly used, which include property graphs, hypergraphs, and triples.

Graph databases do not depend as 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 (Robinson et al., 2015). 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.

2.3 Database Triggers

The database trigger is a technology that is based on ideas that originated in the mid-1970s. It started with the adding of rule languages and rule processing to databases to enhance functionality. In 1973, the CODASYL Data Description Language Committee defined an ON clause, which triggered a database procedure when a data manipulation language operation was performed (Eriksson, 1997). Morgenstern (1983) was the first that referred to databases that used such mechanisms to perform automated view and constraint maintenance as active databases.

Initially, the rule processing research was divided into two separate areas: deductive databases and active databases. In deductive databases, programming rules were used to extend the functionality of the user interface provided by the database query language

(Widom, 1994). In active databases, production rules were used to provide automatic execution of database operations in response to certain events (Widom, 1994).

However, Widom (1994) contended that these two research areas were not actually separate, but rather on opposite ends of a spectrum. This spectrum was created by the level of abstraction of the rule language used. Deductive rules have the greater abstraction, while active rules have the least.

Only in the 1990s did the trigger technology gain traction and make an appearance in some commercial databases. The researchers of that time very descriptively referred to triggers as event-condition-action rules (Simon & Kotz-Dittrich, 1995). These were rules that executed their action when the condition was true after being initiated by a triggering event. The condition was an optional part and rules without conditions were called event-action rules.

This technology was formally added in 1999 to the ISO/IEC 9075 SQL standard and subsequently updated in the 2008 version (ISO/IEC JTC1/SC32, 2011). The name trigger was adopted when it was first incorporated into the SQL standard. Since then, database triggers have become a well-known feature of many relational databases.

Triggers are implemented for variety of reasons to solve many divergent problems. Ceri, Cochrane, and Widom (2000) distinguish between two types of triggers: handcrafted and generated. Handcrafted triggers are written by hand to solve specific application problems. Generated triggers are produced automatically for a specific purpose and parameterised for use in various applications. The authors suggest that there are different uses for the different types of triggers.

For example, generated triggers are well-suited to maintain constraints and materialised views in a database. The goal of the trigger would be to maintain a particular constraint to guarantee referential integrity. For materialised views, the goal of the trigger would be to keep the particular view consistent with the source data as it changes. Once a constraint or view has been declared, a set of triggers can then be generated automatically from the specification of the constraint or view.

One of the suggested uses for manually crafted triggers is to create internal audit trails. Now, however, developers have leveraged this use for auditing in their own external applications. When used for external auditing, these triggers normally take data from

the original tables and store them in dedicated audit tables. Since these tables are part of the custom application, there is no particular layout or content. But generally the audit tables are able to indicate, which user performed what operations and when the operation was performed.

Independent of usage, all the actions performed by triggers can be classified into two groups: general actions and condition specific actions. General action triggers perform their action every time the triggering operation is performed. This is what makes this group of triggers suitable for performing general auditing. For example, a financial institution needs to keep track of all transactions being performed on their system. They can utilize general action triggers to record relevant information for each and every transaction performed. This information would include the exact nature of the transaction, the time it was performed and who the user was that performed the action.

In contrast, condition specific action triggers only perform their action under specific circumstances. These circumstances can be based on either the data that is being manipulated by the triggering operation or by the user that is performing the triggering operation. For example, financial institutions might be legislated to report a certain type of transaction that exceeds a specified amount. This requirement can be implemented with a condition specific trigger that checks the data of all transactions being performed. This specific trigger only fires when the transaction type matches the type that needs to be reported and the transaction amount exceeds the legislated amount.

Consider the situation where the financial institution might, for security reasons only, allow supervisors to perform certain transactions in their system. However, due to the volume of these types of transactions and the ratio of supervisors to normal operators, it is not possible to have only the supervisors perform those transactions. Therefore, they might have implemented an override that the normal operator can invoke, that allows him to also perform these restricted transactions after a supervisor has authorised the transaction with his credentials.

Naturally, the financial institution wants to keep a very close eye on those override transaction to spot any irregularities as soon as possible. A condition specific action trigger can be used here to separately audit these restricted transactions. The trigger is specified to only fire when a user, that is not part of the supervisor group on the system,

performs any of the restricted transactions.

The discussion of database triggers thus far is sufficient for introductory purposes. Chapter 5 will continue this discussion with a look at trigger implementations. This also concludes the introduction of database systems, the first key element of this dissertation.

2.4 Conclusion

This chapter started with providing an overview of some of the core aspects that form the foundation of this dissertation. The first key element of this dissertation is the database system. This chapter introduced database systems with a discussion of the historical development of the major data models. Focus was then placed on relational and NoSQL databases with emphasis on NoSQL databases. Finally, the concept of database triggers was presented.

The next key element of this dissertation is the field of digital forensics. The specific area of digital forensics that relates to the database system is called database forensics. The next chapter provides a general overview of the field of digital forensics and a more detailed discussion on the area of database forensics. The concept of attribution, and how it relates to digital forensics, is also explored in this next chapter.

Chapter 3

Digital Forensic Science Overview

This chapter continues with the overview of some of the core systems and aspects that underpin this dissertation. Another key element of this dissertation is the field of digital forensics. Of particular interest are certain processes and techniques that are used to perform forensic examinations.

Section 3.1 examines the field of forensic science and digital forensics. Different forensic processes and approaches are presented and discussed. Specific focus is placed on database forensics to explore how it relates to general digital forensics. In section 3.2, attention is given to the concept of attribution. Specifically, the process of forensic attribution is explored in more detail. Section 3.3 concludes this chapter.

3.1 Forensics

Forensic science, or simply forensics, is now widely used by law enforcement to aid them in their investigations of crimes committed. Forensic science technicians, which are specifically trained law enforcement officials, perform a number of forensically sound steps in the execution of their duties. These steps include the identification, collection, preservation and analysis of physical artefacts and the reporting of results. One critical part is the collection and preservation of physical artefacts. The collection needs to be performed in such a manner that the artefacts are not contaminated. The artefacts then need to be preserved in such a way that their integrity is maintained. The reason why

this part is so critical is so that any evidence gained from the analysis of these artefacts cannot be contested. The evidence found would be used to either implicate or exonerate any involved parties. Any doubt about the integrity of the artefacts collected could lead to the evidence being dismissed or excluded from legal proceedings.

3.1.1 Digital Forensics

In digital forensics these steps are more commonly referred to as processes. There have been a number of process models developed to guide the digital forensic examiner (Pollitt, 2007). The digital forensic process that matches the collection and preservation step in the physical world is the acquisition process. Traditionally, this process involves the making of exact digital copies of all relevant data media identified (Adelstein, 2006). However, database forensics needs to be performed on information systems that are becoming increasingly complex. Several factors influence the way that data is forensically acquired and how databases are analysed. They include data context, business continuity, storage architecture, storage size and database models.

Historically, digital forensics attempts to collect and preserve data media in a static state, which is referred to as dead acquisition (Adelstein, 2006). Typically, this process starts with isolating any device that is interacting with a data medium by disconnecting it from all networks and power sources. The data medium is then disconnected or removed from the device and connected via a write-blocker to a forensic workstation. The write-blocker ensures that the data medium cannot be contaminated while being connected to the forensic workstation. Software is then used to copy the contents to a similar medium or to an alternative medium with enough capacity. Hashing is also performed on the original content with a hash algorithm, such as the Message Digest 5 (MD5) or Secure Hash Algorithm 1 (SHA-1) (Adelstein, 2006). The hashes are used to confirm that the copies made are exact copies of the originals and have not been altered. The hashes are also used throughout the analysis process to confirm the integrity of the data being examined. Once the copies have been made, there is no more need for the preservation of the originals (Cohen, 2009). However, if the data being examined is to be used to gather evidence in legal proceedings, some jurisdictions may require that the originals are still available.

A different approach is to perform live acquisition. This involves the collection and preservation of both volatile data (e.g. CPU cache, RAM, network connections) and non-volatile data (e.g. data files, control files, log files). Since the acquisition is performed while the system is running, there are some risks that affect the reliability of the acquired data. These risks, however, can be mitigated by employing certain countermeasures (Carrier, 2006).

In current information systems there are several instances where it has become necessary to perform live acquisition. Firstly, in a permanently switched-on and connected world, the context around the imaged data may be required to perform the forensic analysis. This includes volatile items such a running processes, process memory, network connections and logged on users (Adelstein, 2006). One area where the context gained from live acquisition is particularly useful is when dealing with possibly encrypted data. This is because the encrypted data might already be open on a running system and the encryption keys used cached in memory (Hargreaves & Chivers, 2008). The increasing prevalence of encryption usage to protect data by both individuals and organisations increases the need for more live acquisitions to be performed.

Another instance where live acquisition is performed is when business continuity is required. For many organisations, information systems have become a critical part of their operations. The seizure or downtime of such information systems would lead to great financial losses and damaged reputations. The shutdown of mission critical systems might even endanger human life. During forensic investigations, such important information systems can thus no longer be shut down to perform imaging in the traditional way (Adelstein, 2006).

The complex storage architecture of current information systems also necessitates the use of live acquisition techniques. To ensure availability, redundancy, capacity and performance, single storage disks are no longer used for important applications and databases. At the very least, a redundant array of independent disks (RAID) or a full blown storage area network (SAN) is used. Both of these technologies group a variable number of physical storage disks together using different methodologies. They present a logical storage disk to the operating system that is accessible on the block-level.

In such a storage configuration, a write-blocker can no longer be efficiently used.

There simply may be too many disks in the RAID configuration to make it cost and time effective to image them all (Adelstein, 2006). In the case of a SAN, the actual physical disks holding the particular logical disk might not be known, or might be shared among multiple logical disks. These other logical disks may form part of other systems that are unrelated to the application or database system and should preferably not be affected. Attaching the disks in a RAID configuration to another controller with the same configuration can make the data appear corrupt and impossible to access. RAID controller and server manufacturers only support RAID migration between specific hardware families and firmware versions. The same would hold true for the imaged disks as well.

While it is still technically possible to image the logical disk the same way as a physical disk, it may not be feasible to do so either. Firstly, the size of the logical disk may be bigger than the disk capacity available to the forensic examiner (Garfinkel, 2010). Secondly, the logical disk may hold a considerable amount of other unrelated data, especially in a virtualised environment. Lastly, organisations may be running a huge single application or database server containing many different applications and databases. Due to hardware, electricity and licensing costs, the organisation may prefer this to having multiple smaller application or database servers.

3.1.2 Database Forensics

Database systems have their own complexities that affect digital forensic examinations. The models used by the database manufacturers are tightly integrated into their database management systems and are many times of a proprietary nature. Reverse engineering is purposely being made difficult to prevent their intellectual property being used by a competitor. Sometimes reverse engineering is explicitly prohibited in the licensing agreements of the usage of the DBMSs. To forensically analyse the raw data directly is thus not very easy, cost-effective or always possible. The data also needs to be analysed in conjunction with the metadata because the metadata not only describes how to interpret the data, but can also influence the actual seen information (Olivier, 2009). The usage of the DBMS itself, and by extension the model it contains, has become the necessary approach to forensically analyse databases.

The database analysis can be performed in two ways: an analysis on site or an analysis in a clean laboratory environment. On site the analysis is performed on the actual system running the database. In the laboratory, a clean copy of the DBMS with the exact same model as used in the original system is used to analyse the data and metadata acquired (Olivier, 2009). Both ways can be categorised as live analysis due to being performed on a running system. In the first instance, the real system is used, while in the second, a resuscitated system in a more controlled environment is used (e.g. single user, no network connection).

Due to all these complexities associated with applications and particularly databases, live acquisition is the favoured approach when dealing with an information system of a particular size and importance. Fowler documents such a live acquisition in a real world forensic investigation he performed on a Microsoft SQL Server 2005 database (Fowler, 2007a). It should be noted that both the operating system and the DBMS are used to access and acquire data after being authenticated. To preserve the integrity of the acquired data, he uses his own clean tools that are stored on a read-only medium (Carrier, 2006). However, the mere accessing of the system will already cause changes to the data, thus effectively contaminating it before it can be copied. Since all the operations performed during the acquisition are documented, they can be accounted for during a subsequent analysis. Hence, this type of contamination is acceptable as it can be negated during analysis.

3.2 Attribution

Having completed the introduction of the field of digital forensics, focus is now placed on attribution. First, an overview of the field is given to provide context for the remainder of this dissertation. Forensic attribution processes and techniques are then discussed and their implications on database forensics explored.

3.2.1 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, for example, 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 (Kelley, 1973). 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) (Wallenius, Mayer, & Ray, 2006). Authorship attribution refers to the process of inferring characteristics of the author from the characteristics of documents written by that author (Juola, 2008).

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" (Wheeler & Larsen, 2003). They define the resulting identity as a person's 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 Internet Protocol (IP) address or Media Access Control (MAC) address.

Boebert (2010) breaks the attribution question down into two attribution problems: technical attribution and human attribution. 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 that different types are useful in different contexts (Clark & Landau, 2010). 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. distributed denial of service (DDoS) attack), 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 counteracted. 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 (Clark & Landau, 2010).

3.2.2 Forensic Attribution

When attribution is done as part of an examination using scientific methods, the term forensic attribution is used. Forensic attribution is performed during the digital evidence interpretation step in a forensic examination (ISO/IEC JTC1/SC27, 2015). This step is part of the investigative processes as defined in the ISO/IEC 27043 standard that describes incident investigation principles and processes (ISO/IEC JTC1/SC27, 2015).

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 (Cohen, 2010). 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 (Shamsi, Zeadally, Sheikh, & Flowers, 2016). 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" (Rid & McBurney, 2012).

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 (Clark & Landau, 2010). 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 3.1: Attribution Processes Summary

Author(s)	**Digital Realm**	**Physical Realm**
Cohen (2010)	• Identify closest computer • Identify initiating computer	• Identify individual behind initiating computer • Identify organisation behind individual
Shamsi et al. (2016)	• Identify cyber weapon	• Identify country or city • Identify person or organisation
Clark and Landau (2010)	• Identify computer	• Identify individual • Identify organisation

Table 3.1 summarises these different attribution processes. Even though the authors use different terminology to describe the parts of their processes (steps/levels/categories), the parts and their goals appear to be very similar. For the sake of clarity, this dissertation 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 type of attribution may not be always be needed. The dissertation 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 Larsen (2003). According to them, no single technique can replace all others and a combination of techniques can help compensate for their respective weaknesses.

In order to access and operate many digital systems, authentication is required. The authentication mechanism is used to identify users and grant the corresponding access to the system. The access available to a particular user depends on the authorisation given to the user credentials. Unauthorised users are prevented from accessing and operating the system.

Thus, one of the techniques that the forensic examiner can employ is to make inferences based on the authentication and authorisation information (Cohen, 2009) found in a digital system. This information can enable him to create a basis for attribution as follows: the authentication information provides the actors of the digital system, while the authorisation information gives the actions that these actors can perform (Hauger & Olivier, 2015a).

Many digital systems create and maintain audit records and metadata of different sorts (Cohen, 2009). They include date, timestamps and ownership of files, authentication and program execution logs, output files generated as part of the execution of programs, network traffic logs and assorted other traces. These records are created as part of the normal operation of the system. They allow for the confirmation and review of activities, as well as for debugging in the case of errors.

Due to this circumstance, another attribution technique that can be employed, is to order and connect these different traces that can be found to build a chain of events (Cohen, 2009). The sequence of these events will describe how the system arrived at the

current state. By going further and determining the actions that led to the events and the actors that performed these actions, the person or program responsible can possibly be identified (Hauger & Olivier, 2015a).

However, both the authentication information and the various traces are not infallible. Attackers can bypass authentication or steal the credentials of an innocent user. Traces can many times be easily modified or even removed to hide malicious activity. Furthermore, these attribution techniques can be abused by malicious actors. They can intentionally use user credentials and create traces that falsely attribute actions to innocent third parties.

It is thus important for the forensic examiner to attempt to obtain multiple independent traces that portray the same event. These independent traces can then be used to correlate the actions and identify any manipulation. By either excluding the manipulated information or reconstructing the original information, the forensic examiner can move closer to identifying the real actor.

Performing forensic attribution in relational databases was investigated by Olivier (2009). 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 is built into many relational databases (Hauger & Olivier, 2015a). The same caveats as in general digital forensics also apply to database forensics.

3.3 Conclusion

This chapter continued with the overview of the core aspects that form the foundation of this dissertation. First, the concept of digital forensics was presented with the spotlight on database forensics. The concept of attribution was then introduced with specific focus on forensic attribution.

Since the field of database forensics is still relatively new, the amount of scientific research in that field is limited. Instead of a traditional literature review, the next chapter presents a survey of all major database forensic research conducted in a nine

year period from 2009 to 2017. It is a follow-up to a previous survey conducted in 2009. The chapter also attempts to determine why research in this area has been lacking so far.

Chapter 4

Database Forensics Research

This chapter takes a different approach to the traditional literature review, which has a limited amount of related research material available to review. This perceived lack of related research, and scientific research in database forensics in general, is echoed in some of the recent research papers on database forensics. Different database forensic researchers have also implied that not enough research is being conducted in this important field (Fasan & Olivier, 2012; Pieterse & Olivier, 2012).

Therefore, this chapter investigates the claim of lacking research in the field of database forensics as expressed by these researchers. Scientific material on digital forensics published in the last nine years is surveyed. In order to be able to quantify the research into database forensics, research into the currently active field of cloud forensics is also surveyed. Similar to database forensics, cloud forensics is a relatively new sub discipline of forensic research and it does not have any significant material published before 2009. These inherent similarities make it a fitting benchmark for comparison purposes. Various sources are consulted ranging from important forensic journals and conference proceedings to academic search engines.

This chapter also attempts to establish reasons as to why there could be a lack of research in the field of database forensics. To enable a systematic approach of finding possible reasons, the research identified in the survey that was conducted in the field of cloud forensics is analysed. The reasons given by the authors for performing the cloud forensic research are identified and their work is broadly categorised. Parallels are then

drawn to the field of database forensics. This chapter extends on work from a previously published paper (Hauger & Olivier, 2015c).

Section 4.1 provides some context about the nature of databases and database forensic research. Section 4.2 surveys the digital forensic scientific literature of the past nine years. Section 4.3 analyses the results of the survey. Thereafter, section 4.4 discusses some reasons for research and publication or the lack thereof in digital forensics. Section 4.5 concludes this chapter.

4.1 Research Classification

In this section some context is provided with regards to the different types of database forensic research being conducted. The nature of databases is also examined to ascertain how this determines the classification of the surveyed research as database forensic research.

The forensic researchers working on databases appear to follow two different approaches. The first approach contends that a database is actually nothing more than files that reside inside a file system on a storage medium. Some files are the container for the database data and metadata, whilst other files are the software that runs the database. This means databases can be analysed for clues similar to other important software, such as email and web browser software (Chivers & Hargreaves, 2011; Pereira, 2009). This view places database forensics as a sub-discipline of file system forensics. The same techniques, such as imaging and file carving, are used.

The other approach contents that databases are much more complex than plain files. Databases have multiple dimensions that are interconnected and need to be analysed together to provide an accurate depiction of the truth. Olivier advocated this approach in his 2009 paper "On metadata context in Database Forensics" (Olivier, 2009). He identified four layers that need to be considered: the data model, the data dictionary, the application schema and the application data. After the integrity of each layer has been established, the database management system (DBMS) itself can then be used to perform a live forensic analysis.

The approach used to forensically analyse databases also appears to define the type

of research conducted. The group that treats databases as files builds on the file forensic discipline and the scientific research already done in that area. It produces incremental research that specialises the existing scientific methodologies and knowledge for the forensic analysis of database files.

The group that utilises the DBMS for the analysis of running databases performs new scientific research in the area of forensics. The research is new because live forensic analysis itself is a more recent technique that is being employed out of necessity. Furthermore, the live analysis of databases has to deal with the complexity of interconnected layers, making it distinctive.

Olivier raised a number of concerns with the approach of analysing a database as files. The application data is normally carved from the disk image and the application schema is either inferred or reconstructed. The data model and data dictionary are not considered in this process (Olivier, 2009). How can one then be certain that the data is interpreted correctly without the data dictionary? How can one be certain that the data was correctly carved without the data model? What complicates matters even further is that many data models are proprietary and not documented.

Due to these concerns, the information or facts obtained from database file reconstruction should only be used as leads. Should these facts be used as evidence, they might be rightly challenged in a court of law. If the second approach of using the DBMS is used, all four layers will be automatically considered. If the integrity of each layer has been scientifically proven, then this approach can provide evidence that will hold up in a court of law.

In order to be able to classify the forensic research done on database systems and files, a definition of what constitutes a database is required. Since many different types of database systems exist, a single generic definition might not be very useful. A better option would be to define characteristics that make a system a database system. This is exactly what Atkinson et al. did in their paper titled "The Object-Oriented Database System Manifesto" (Atkinson et al., 1989). Their paper presents a number of mandatory characteristics that, according to the authors, define an object-oriented database system.

The authors contend that a system qualifies as an object-oriented database system if it is both a database management system (DBMS) and follows object-oriented

principles. The following characteristics define a DBMS: persistence, secondary storage management, concurrency, recovery and an ad-hoc query facility (Atkinson et al., 1989).

Persistence implies that the data should survive the termination of the process without the user having to make it explicitly persistent. Secondary storage management refers to mechanisms provided by the DBMS to manage very large databases. These mechanisms include index management, data clustering, data buffering, access path selection and query optimisation. These mechanisms work behind the scenes to enhance the performance when the database size becomes large.

Concurrency implies the management of simultaneous users interacting concurrently with the DBMS, possibly manipulating the same data. Features such as atomic operations and serialisation of a sequence of operations are required. Recovery refers to the ability of the DBMS to bring the data back to a coherent state after hardware and software failures. The ad-hoc query facility denotes the provision of a service that allows the user to make elementary queries to the database using some structured language.

Based on this classification the research work of, for example, Chivers and Hargreaves cannot be classified as database forens0ics because the Windows Search Database is not a database (Chivers & Hargreaves, 2011). The work of Pereira, however, could be considered as database forensics because SQLite arguably satisfies the criteria for a database (Pereira, 2009).

4.2 Literature Survey

This section surveys the scientific literature for forensic research conducted in the nine years from 2009 to 2017, and compares the results to a previous survey done by Olivier in 2008 that was published in March 2009 (Olivier, 2009). This same comparison survey was already carried out at the start of 2015 for the first six years (Hauger & Olivier, 2015c). This section will thus compare the current results as well as the 2015 results to the original 2009 survey.

In the 2009 paper, Olivier showed the lack of scientific work around database forensics by searching for published information consulting various sources (Olivier, 2009). He started with the leading digital forensic journals *Digital Investigation* and *International*

Journal of Digital Evidence. Next, he looked at papers presented at the digital forensic conferences International Federation for Information Processing (IFIP) WG 11.9 and Digital Forensics Research Workshop (DFRWS).

He also consulted the digital libraries of the following professional societies: *ACM Digital Library* from the Association for Computing Machinery (ACM) and *IEEEXplore* from the Institute of Electrical and Electronics Engineers (IEEE). The digital library *ScienceDirect* from the scientific publisher Elsevier was consulted as well. He then used the research oriented search engines Google Scholar (scholar.google.com) and Live Search Academic (academic.live.com) to find material on database forensics. He also looked to see how many books were published on the topic of database forensics or addressed the topic by consulting Google Books (books.google.com).

In this survey the exact same searches were repeated some six and nine years later. By comparing the new numbers to those of the original survey, it is possible to determine how the research output has changed over the survey period. However, to gauge the rate of increase and volume of new research material published, it is necessary to compare the numbers to some type of benchmark.

The first choice for a comparison benchmark would be the established general discipline of digital forensics, originally also referred to as "Computer Forensics". The problem with this choice is that the comparison of research output volume would only confirm that database forensics is a particularly sized sub discipline of digital forensics. It would not indicate if the research output in the area of database forensics is actually poor or not.

One can also argue that research trends and output in a more mature area differ from a new and emerging area. A more suitable choice for comparison would thus be a similarly emerging sub discipline of digital forensics. What immediately comes to mind is the currently very active area of digital forensics called cloud forensics. Since the "Cloud" in general is an emerging technology, the forensic science research conducted around it is similarly new and emerging.

The disciplines of database forensics and cloud forensics have inherent similarities. Besides both being sub disciplines of digital forensics, they also have a similar age. Furthermore, there would not have been any significant amount of cloud forensic work

published before 2009, making it a fitting choice for the survey period. Both disciplines also deal with the storage and structured manipulation of data.

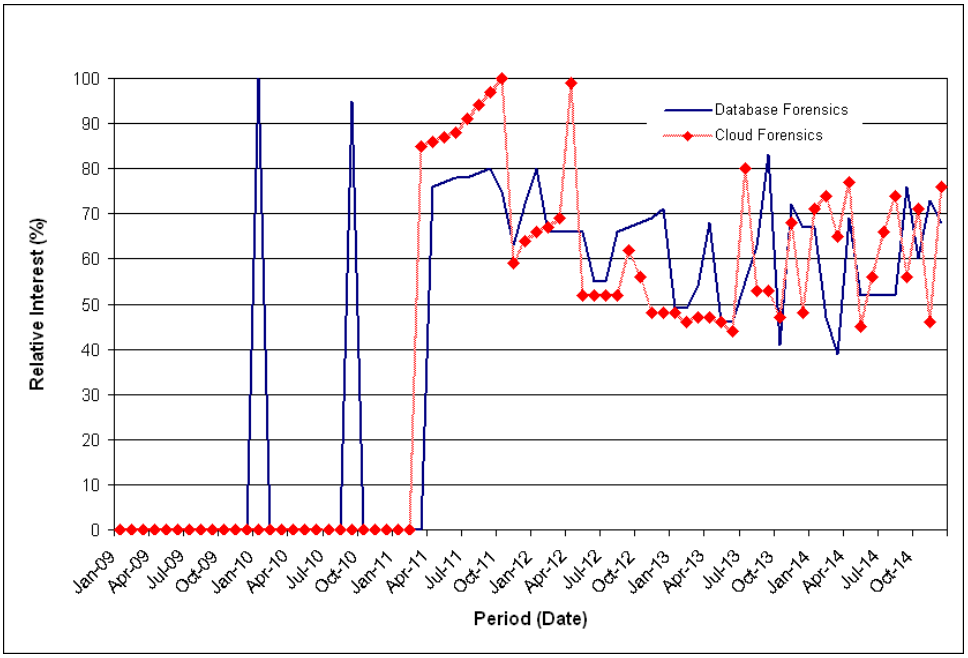

Figure 4.1: Web search interest over 2015 survey time period. Data Source: Google Trends (www.google.com/trends), July 2015

Another striking similarity is the interest of internet users in both disciplines in recent years. Figure 4.1 shows the search trends for Google web searches of the six years from 2009 to 2014 on the topics "Database Forensics" and "Cloud Forensics". The two topics have a remarkably similar graph, but initial interest in cloud forensics appeared somewhat later than the initial interest in database forensics. The graph depicts the relative interest of each topic compared to the total number of web searches performed on Google. Each graph is normalised to a hundred percent. The actual number of searches performed might not necessarily be the same, but the level of interest follows the same pattern.

An attempt to extend figure 4.1 to cover the full survey period failed due to the changes made by Google to the way it samples and uses its search data to create the trends data. Samples are now taken weekly instead of monthly and the included data is a randomly selected subset of the actual search data (Google, 2018). This random

sampling might not have an impact on popular search topics that occur constantly at very high volume, but for less popular search topics that may only occur once a week, this is fatal.

Depending on when the weekly sample for such a less popular search topic is taken, it may show as zero interest or some positive interest. This is manifested in completely changing trend graphs when repeating the exact same trend search on different days or when slightly changing the search period on the same day (e.g. adding or removing a single day).

For completeness sake, a Google trends search on the topics "Database Forensics" and "Cloud Forensics" for the remaining survey period was conducted. Figure 4.2 shows the search trends for Google web searches of the last three years from 2015 to 2017.

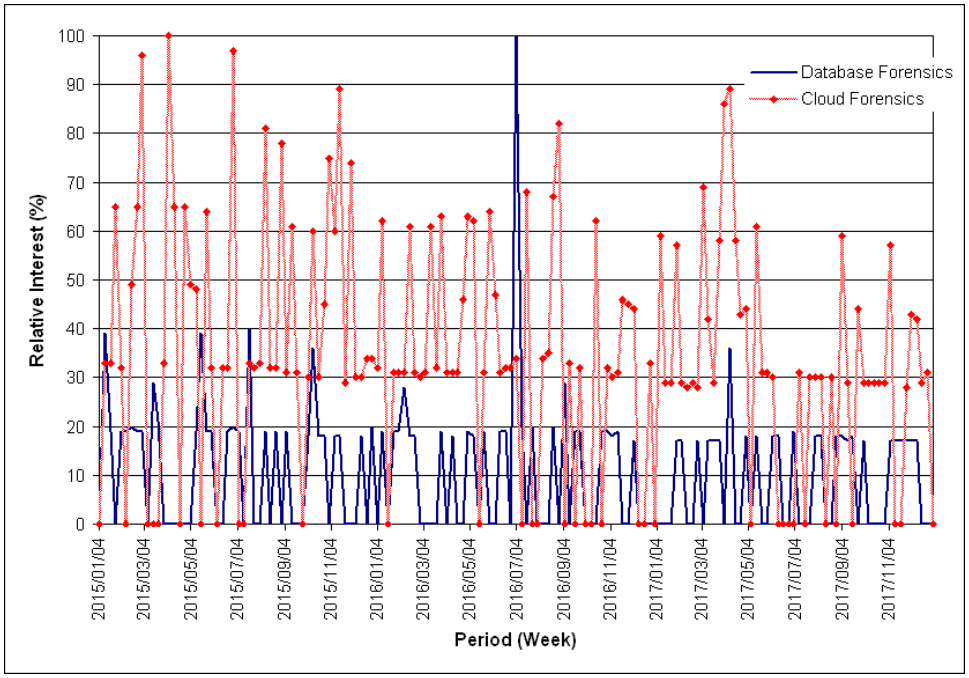

Figure 4.2: Web search interest in the time period between the 2015 survey and the current survey. Data Source: Google Trends (www.google.com/trends), September 2018

Even though the random samples for both topics are taken at the same point in time, an absolute comparison between them is meaningless. When the number of web searches in a sample falls below a certain threshold, it is represented as zero. This creates

artificially big differences in the interest between the two topics at such points and leads to an unnaturally moving graph.

However, what is evident in figure 4.2 is that the average interest in cloud forensics is noticeably higher than in database forensics over the same time period. The accompanying averages provided by Google put the database forensics average at ten percent. The average for cloud forensics is put twice as high at twenty percent.

The research oriented search engine from Microsoft (academic.live.com), which the 2009 paper also consulted, no longer exists. There appears to be a replacement search engine called Microsoft Academic (MA). In 2015, this search engine was still in beta and located at academic.research.microsoft.com. Since 2017, this search engine is in version 2.0 and located at academic.microsoft.com. The first assumption, based solely on the name, is that this search engine only covers research done or funded by Microsoft. However, closer inspection revealed that it does actually cover all the research and was thus used as a replacement.

The survey that was performed followed the same consultation order as the 2009 paper. The survey started with consulting the same digital forensic journals as the 2009 paper. All issues published since March 2009 were studied. The *International Journal of Digital Evidence* published its last issue in the fall of 2007. There was therefore no need to search this journal for new material. The journal *Digital Investigation*, however, has continued to publish regularly. In all subsequent issues published since the issues that were contained in the original survey, there are exactly three articles on database forensics. In contrast, there were twelve articles published on cloud forensics in the same issues.

The survey continued with consulting the proceedings of the same digital forensic conferences as the 2009 paper. The proceedings of the conferences held since the March 2009 were studied. Revised and heavily edited versions of the papers presented at the IFIP WG 11.9 conferences are published in a series of books titled *Advances in Digital Forensics*. The volumes V (2009) to XIII (2017) were consulted. A total of six papers were found on the topic of database forensics, while ten papers dealing with cloud forensics were published.

The DFRWS conference keeps an archive of the proceedings on its website (dfrws.org).

The archives from DFRWS 2009 to DFRWS 2017 including the European conference DFRWS EU, which started in 2014, were consulted. Only four papers pertained to cloud forensics, while three recent papers addressed database forensics. A workshop on OpenStack Cloud Forensics was also presented.

Subsequently the quoted phrases "Database Forensics" and "Cloud Forensics" were used on the various digital libraries and search engines. IEEEXplore now returns 14 matches for database forensics, where originally there were no matches found. For cloud forensics, however, IEEEXplore returns 77 matches. ScienceDirect searches still ignore the quotation marks even though their own help on searching for phrases instructs one to use double quotation marks. It returns 82 matches, but many of them refer to DNA forensics in the medical field. If one limits the field to computer science only, there are 22 matches left, of which around 10 relate specifically to database forensics. Cloud forensics, on the other hand, produces 50 matches, of which just more than half are relevant.

In 2015, the ACM Digital Library appeared to be not only searching ACM published content, but also included content from the IEEE and other scientific publishers such as Elsevier and Springer. Database forensics then produced 19 matches, some of which were books about database security. Cloud forensics produced 45 matches, which contained some articles about social networking forensics. Somewhere in the last three years, the ACM updated their search portal and now, by default, the portal only searches their own content. Unfortunately, this updated portal appears to only have material available from 2012 onwards. Database forensics now only produces four matches, while cloud forensics produces eleven matches. Due to changes in the sources as well as the period of material searched by the ACM portal, any comparison to previous data is meaningless.

What is quite evident from all the matches found by the various search platforms for the phrase "Cloud Forensics" is that there was nothing published on cloud forensics before 2010. This supports the choice of using the area of cloud forensics for comparative purposes.

Finally, attention was given to the research oriented search engines. Google Scholar now produces 426 hits for the phrase "Database Forensics". Due to the bigger number of hits, the relevance of all hits has not yet been established. Microsoft Academic produces 336 hits which seems to include DNA databases. After excluding the DNA related

material, only 156 hits remain. In comparison there are 1640 hits for "Cloud Forensics" on Google Scholar and 169 hits on Microsoft Academic.

The Google Books search engine was also consulted in the 2009 paper. Searches with the same phrases as used before were repeated. For comparative purposes, the phrase "Cloud Forensics" was also entered. Google Books now produces 1340 hits for the phrase "Database Forensics", while the phrase "Cloud Forensics" brings back 2560 hits. Hits for the phrases "Digital Forensics" and "Computer Forensics" have shot up to 54600 and 60600 respectively.

4.3 Discussion

In this section the results from the survey examined in the previous section are discussed using three comparative tables.

Table 4.1: 2009 vs. 2015 vs. Current Survey (Data to the end of the previous year for each survey where possible)

Source	**Counts**		
	'09	*'15*	*'18*
Digital Investigation	0	1	3
Int. Journal of Digital Evidence	0	0	0
IFIP WG 11.9	0	4	6
DFRWS	0	0	3
IEEEXplore	0	7	14
ScienceDirect	1	10	10
ACM	0	19	4
Google Scholar	12	236	426
Microsoft Academic	0	5	156

Table 4.1 compares the results from 2009 survey to the survey performed in the 2015

paper and to the current survey for the topic of database forensics. All sources show an increase in published material in the current survey.

Initially, Google Scholar and the ACM showed the highest increases. Both these sources consulted a variety of electronic libraries. As already discussed in the previous section, the ACM search functionality used to connect to a number of other publishers and thus had access to a big amount of published material. It was, therefore, not unexpected that the ACM would return a greater amount of matches. However, due to the change by the ACM to search its own material only, the ACM matches have reduced significantly in the current survey.

In the current survey, Microsoft Academic and Google Scholar show the highest increases. Both these academic oriented search engines use their companies' main search engines (Bing and Google respectively) to search the Internet. These search engines will pick up all available electronic publications and libraries exposed on the web. Unfortunately neither Google Scholar nor Microsoft Academic provide a list of libraries included.

It rather appears that Google is trying to be as comprehensive as possible. On the positive side, this means that the libraries of scientific and academic institutions are included, which enables material such as whitepapers, dissertations and theses to be added. On the negative side, this also adds pseudo-science articles from predatory journals which are not peer-reviewed (Beall, 2014). It is therefore not surprising that Google would find the greatest amount of matches.

Google is known to regularly update its search technology. The two biggest changes in the past nine years were the introduction of Google's web indexing system *Caffeine* and the conversational search update (Grimes, 2010; Singhal, 2013). How those changes have affected the Google Scholar and Books search services is not known. However, such changes are not expected to have a significant impact on this research.

Table 4.2 compares the current survey results for database forensics with those for cloud forensics. The results produced by the Microsoft academic search engine are not in line with the trend shown by all other sources. Based on the low amount of matches on both topics compared to the Google search engine, it appears that the Microsoft search engine does not yet have enough material enabled for searching. This might be due to

Table 4.2: Database Forensics vs. Cloud Forensics Prevalence (Data to the end of the previous year for each survey where possible)

Source	**Counts**			
	DB '15	*DB '18*	*Cloud '15*	*Cloud '18*
Digital Investigation	1	3	9	12
IFIP WG 11.9	4	6	6	10
DFRWS	0	3	3	4
IEEEXplore	7	14	29	77
ScienceDirect	10	10	20	30
ACM	19	4	45	11
Google Scholar	236	426	428	1640
Microsoft Academic	5	156	3	169
Total	**282**	**622**	**543**	**1953**

the beta status of the Microsoft search engine. The results from the Microsoft engine will thus not be considered any further in this discussion.

All other sources show a higher publication count for cloud forensics versus database forensics over the same period. The differences vary between the different sources, but all of them show a more than thirty percent higher count. Looking at the total amount of material published for both topics, there are nearly twice as many publications on cloud forensics than on database forensics.

Table 4.3 shows the amount of hits the Google Books search engine found for the various search phrases. It compares the number of hits recorded in the 2009 paper to the survey performed in the 2015 paper and the current survey. Based on the numbers, it would appear that a huge amount of books were published on the various topics in the last nine years. However, one has to keep in mind that Google Books will include books in its results even if they contain the various phases just once in the entire text. That means books from other disciplines that simply reference forensics in some way, will also

Table 4.3: Google Books 2009 vs. 2015 vs. Current Survey (Data to the end of the previous year for each survey where possible)

Phrase	**Counts**		
	'09	*'15*	*'18*
Database Forensics	1	284	1340
Cloud Forensics	–	291	2560
Digital Forensics	188	9510	54600
Computer Forensics	716	33200	60600

be counted.

Another factor that can also influence the increase of books, is that Google has steadily been adding older books to its library. Those would include books published before 2009, which would not have been counted in the 2009 paper. What is interesting, however, is that the amount of books referencing database forensics and cloud forensics were similar in 2015. This means that, at that time, the awareness of both disciplines in the literary community was still the same. Since then the books have followed the same trend as all the other material and there are now twice as many books referencing cloud forensics than database forensics.

The size of the sample for the survey might appear to be too small to be able to make definite conclusions. However, given the specific sources that were used, one is arguably dealing with a big enough portion of the entire population of peer-reviewed digital forensic research material.

4.4 Possible Explanations

This section contemplates some possible reasons for the slow pace of scientific research in database forensics compared to other digital forensic areas, such as cloud forensics.

Attempting to explain the absence of database forensics research in the absence of research papers in this area is quite difficult. An alternate approach would be to analyse

research papers published in the same period in a different and more prolific research area. Such an analysis could establish motivations and objectives that might be different or not applicable to database forensics. Thus, the cloud forensic papers identified during the survey were analysed to gain insight into the absence of database forensic research.

In order to establish motivations and objectives in a more structured manner, a number of the identified research papers into cloud forensics were chosen to be analysed in greater detail. They are the twelve papers from the *Digital Investigation* journal, the four papers from the DFRWS conference as well as the ten papers from the IFIP conference.

These specific papers were chosen because they focus exclusively on digital forensics and have been peer-reviewed. Using these criteria ensures that the papers are relevant and represent quality research. The abstract and introduction of a conference paper or journal article normally follows a specific template. The template form prescribes that both should contain a problem statement and a motivation for the research. Hence, the abstract of each paper was consulted to establish what problem was being addressed and the motivation. In the cases where the abstract was too abstract and did not follow the template form, the introduction was also consulted to clarify the problem and motivation.

A significant amount of the selected papers deal with analysing how forensics can be performed in the cloud environment utilising current scientific forensic methods and processes. They identify challenges and propose solutions by adapting either the forensic methods and processes, or the cloud infrastructure and processes. Hence, they are evaluating the forensic readiness of the community to handle cloud environment investigations. Nine of the studied papers can be classified in such a category (Delport & Olivier, 2012; Martini & Choo, 2012; O'Shaughnessy & Keane, 2013; Pichan, Lazarescu, & Soh, 2015; Ruan, Carthy, Kechadi, & Baggili, 2013; Ruan, Carthy, Kechadi, & Crosbie, 2011; Ruan, James, Carthy, & Kechadi, 2012; Trenwith & Venter, 2015; Zawoad & Hasan, 2015).

The remainder of the chosen papers, with the exception of three, can be divided equally into two groups. The first group of papers investigates how to determine that cloud systems and platforms were indeed utilised, and how to identify the specific cloud service providers from their artefacts. They also discuss how to extract the necessary

information from these artefacts to allow the retrieval of data from various cloud systems and platforms. Seven of the papers deal with artefacts from specific cloud services (Chung, Parka, Lee, & Kang, 2012; Hale, 2013; Liu, Singhal, & Wijesekera, 2017; Martini & Choo, 2013; Quick & Choo, 2013a; Ras & Olivier, 2012; Roussev & McCulley, 2016).

The second group of the remainder of papers proposes different methods and create different tools to extract data and information from various cloud systems and platforms. Some of them also investigate how forensically sound the retrieved data and information is. A total of eight papers can be put into this category (Dykstra & Sherman, 2012, 2013; Federici, 2014; Leimich, Harrison, & J.Buchanan, 2016; Oestreicher, 2014; Quick & Choo, 2013b; Roussev, Ahmed, Barreto, McCulley, & Shanmughan, 2016; Roussev, Barreto, & Ahmed, 2016).

One of the exception papers argues that the general availability of cloud computing provides an opportunity to help with the automated processing of huge amounts of forensic data (Bhoedjang et al., 2012). Some of the other papers also hint at using cloud computing to provide forensics-as-a-service.

The other exception paper deals with identifying malware and other malicious code in virtualised cloud systems. The paper proposes and tests a specific method, to find and block the calls made by such malicious code (Ahmed, Zoranic, Javaid, III, & Roussev, 2013).

Thus, the purpose for all 26 papers can be divided into five broad categories: cloud forensic readiness, cloud artefact analysis, cloud data acquisition, forensics-as-a-service and cloud security. The last two categories do not pertain directly to performing forensics on cloud systems and are thus not discussed any further.

Databases never had the same challenges to be forensically ready as cloud systems. They were built from the ground up with standard features such as authentication, authorisation and auditing capabilities (Ramakrishnan & Gehrke, 2003). These traces are stored as metadata inside the database and, depending on configuration, also externally in various log files. They make it possible to forensically trace and attribute all events that occur in a database.

Artefact analysis also does not play such an important role in databases. The identification of specific databases has never been a great forensic challenge. Usually, there is

some client or server DBMS software installed that identifies the database. Even if the DBMS software is no longer present or not available, most database files can be easily identified with the help of magic numbers (Kessler, 2014).

Version upgrades or multiple installations could create complications in database identification, but they are not insurmountable. The artefacts of databases that are forensically interesting, are the various log files. There already has been quite a bit of research done in this area (Adedayo & Olivier, 2014; Fruhwirt, Kieseberg, Schrittwieser, Huber, & Weippl, 2012).

Forensic data acquisition methods and processes were being developed, when databases were already part of computer systems. Thus, these methods and processes already indirectly addressed databases. The huge increase in the size of databases, however, has forced forensic examiners to start using live acquisition, and live analysis methods and processes. Some of these processes have already been researched and published (Fowler, 2007a).

In summary, the problem areas addressed by the analysed cloud forensic research are either not relevant to databases or have already been addressed. This would explain why researchers working in these particular forensic fields are not interested in databases.

As noted earlier, the motives for the forensic research where also explored. From the various motivations given by the cloud forensic researchers for doing their research, a number of reasons could be identified. Firstly, the researchers realised that cloud computing systems and storage services (CCSSS) brought new challenges to performing forensic investigations. They also recognised the fast adoption rate of these emerging technologies and the need for the forensic community to keep pace with these developments.

From a forensic point of view, CCSSS can be regarded as disruptive technology. The struggle that the forensic community has, is that these CCSSSs have distributed physical structures and logical processes and that the conventional proven forensic acquisition and analysis methods and processes cannot be easily applied any more, if at all.

A third of the selected papers focused on forensic readiness to address these challenges. This research followed two different approaches. The first group of researchers investigated how one could adapt and enhance current forensic methods and processes

to deal with cloud forensics. The second group researched how one could structure or modify CCSSSs to add the features that would allow the proven forensic concepts and ideas to be applicable again.

The reality, however, is that many of these CCSSSs currently in use, were not designed with forensics in mind at all. They rather focused on the performance, scalability, cost and elegance of the solution. This is in contrast with database systems that were built from the ground up with access control and auditing abilities. These abilities make it possible to forensically trace all events occurring in a database.

This could be compared to having a conference venue that is rented out for events. Items such as the event organisers, occupation dates and payment information are recorded by the venue owner. However, the people attending the event or the actual happenings during the event are not meticulously recorded by the venue owner.

Secondly, the researchers recognised the value of the information contained in the CCSSS for forensic purposes. Forensic practitioners need new methods and tools to acquire and analyse data from the current plethora of cloud platforms now available.

A majority of the researchers also felt that CCSSS would create an increase in cybercrime, as well as produce new forms of cybercrime. Firstly, they provide criminals with new targets to attack and compromise and secondly, they provide powerful platforms to use for their current and future criminal activities.

A recent example of the malicious use of cloud computing systems, was the rental of a few thousand Google cloud computing instances by a hacker group calling themselves the LizardSquad. They used the instances to build a botnet to help them perform DDoS attacks (Krebs, 2015). Another example was the attack on the iCloud storage service platform from Apple. The attackers appeared to have targeted the accounts of arbitrary users and stolen the stored data including the private photos of various celebrities (Higgins, 2014).

One final reason that can never be dismissed as research motivation, is the newness factor of a specific subject. CCSSS is currently a new technology that promises new and revolutionary applications. We are probably still somewhere on the rise section of the hype cycle when it comes to CCSSS. The word “Cloud” has become a buzz word that fills the technical media and has already made it into the main stream media (Mullich,

2011). This omni-present hype creates a pervasive idea in people's minds. So when the time comes to choose a topic for new research, cloud research will quickly present itself.

In contrast, databases represent old technology that has been around for many years. They have matured to the point that they form a natural part of many current computer systems. There have been some recent new developments in the field such as in-memory databases and NoSQL databases. These developments will probably attract some new forensic research as they become more widely used.

A deduction that can thus be made, is that the driving forces for the two research areas are different. Various forces drive the research in the area of cloud forensics, but these same forces are not present with database forensics. The research in the area of database forensics was not at a standstill in the past six years. In fact, the research increased during that period, indicating that there are still other driving forces.

4.5 Conclusion

Instead of a traditional literature review of related research material, this chapter performed a literature survey of all major digital forensic research published since 2009 in the areas of database forensics and cloud forensics. This was compared to the state of affairs in a previous survey published in 2009.

The literature survey indicated that the speed of research into database forensics has increased since the 2009 survey was published. There is quite a bit more published information available than before. However, other newer areas, such as cloud forensics, have produced twice the amount of new research in the same time period. Based on the book analysis of the survey, at least the interest and awareness around both disciplines appears to be the same.

The analysis of the motivations for performing research in the cloud forensics domain has not identified a specific driving force. Rather, a number of different factors have influenced the researchers. These same factors mostly do not apply to the database forensics domain, explaining why these forensic researchers have given little attention to databases. Those researchers that did perform database forensic research were thus driven by different forces, probably inspired by the call for more research in the 2009

research paper.

The next chapter starts working towards answering the initial research question. To this end, it performs an in-depth investigation of database triggers and their workings. It also analyses the interference potential during a forensic examination.

Chapter 5

Database Trigger Implementations

Chapter 2 introduced the database trigger by providing a brief history and their purpose in relational databases. This chapter takes a more in-depth look at triggers and their workings. The different types of triggers that are defined in the standard, as well as other non-standard triggers, are examined. Specifically, this chapter investigates the database triggers as they have been implemented by a chosen number of DBMSs. This chapter is based on work from a previously published paper (Hauger & Olivier, 2015b).

Firstly, section 5.1 provides the context and database selection against which the database triggers will be investigated. To start the investigation, section 5.2 first examines what types of triggers are defined in the standard. Then, section 5.3 inspects the implementations of the standard triggers by the selected DBMSs. Thereafter, section 5.4 looks at other non-standard triggers that some DBMSs have implemented. For each type of trigger the question was asked as to how the usage of that particular trigger could impact the forensic process or method. To conclude the investigation, the database objects that triggers can be applied to, are examined in section 5.5. Section 5.6 concludes this chapter.

5.1 Investigation Context

In chapter 2, it was established that database triggers are designed to perform automatic actions based on events that occur in a database. This implies that there is a wide variety

of commission and omission actions that can be performed by triggers. These actions can potentially have an effect on data inside and outside of the DBMS. Thus, triggers, and the actions they perform, are forensically important. This was already recognised by Khanuja and Adane (2012) in a framework for database forensic analysis they proposed.

The effect that triggers can have on data raises the concern that they could compromise the integrity of the data being examined. Could triggers, due to their nature in combination with the way databases are forensically analysed, lead to the contamination of the data that is being analysed? This chapter attempts to establish if these concerns around triggers are justified.

In chapter 2, it was ascertained that the database trigger is defined in the ISO/IEC 9075 SQL standard (ISO/IEC JTC1/SC32, 2011). This specification could thus be examined to determine, on a theoretical basis, if there is reason for concern. However, the standard is merely used as a guideline by DBMS manufacturers and there is no requirement to conform to the standard. Certain manufacturers also use feature engineering to gain a competitive advantage in the marketplace (Turner, Fuggetta, Lavazza, & Wolf, 1999). They might implement additional triggers based on actual feature requests from high profile clients. Standard triggers might also be enhanced, or other additional triggers implemented, based on the perceived usefulness by the manufacturers. These features could be used to overcome certain limitations in their DBMS implementations. It is therefore necessary to study actual trigger implementations, rather than the standard itself.

There are thousands of database implementations available and to investigate the trigger implementations of all those databases that use triggers would be prohibitive. Thus, the database trigger implementations of a number of proprietary and open-source DBMSs were chosen. The DBMSs investigated were Oracle, Microsoft SQL Server, Mysql, PostgreSQL, DB2, SyBase and SQLite. These selected relational database management systems (RDBMS) are widely adopted in the industry. SQLite is particularly interesting since it is not a conventional database. SQLite has no own server or running process, but is rather a single file that is accessed via libraries in the application using it. SQLite is being promoted as a file replacement for local information storage. Some well known applications, such as Adobe Reader, Adobe Integrated Runtime (AIR), Firefox

and Thunderbird, use SQLite for information storage. SQLite is also very compact and thus well suited for use in embedded and mobile devices. Mobile operating systems iOS and Android make use of SQLite (SQLite Cons., 2014a, 2014c).

The dominance of the selected RDBMSs in the market means that they would be encountered fairly often by the general digital forensic examiner. These RDBMSs are also the most popular based on the number of web pages on the Internet according to solid IT's ranking method (DB-Engines, 2014). The official documentation of these RDBMSs was used to study their trigger implementations. The latest published version of the documentation was retrieved from the manufacturer's website (IBM, 2012; Microsoft, 2012a; Oracle, 2009a; Oracle Corp., 2015; PostgreSQL Group, 2013; SQLite Cons., 2014b; Sybase Inc., 2011). At the time of the investigation, the latest versions available were as follows: Oracle 11.2g, Microsoft SQL Server 2012, Oracle Mysql 5.7, PostgreSQL 9.3, IBM DB2 10, Sybase ASE 15.7 and SQLite 3.8.6.

5.2 Specification

The ISO/IEC 9075 SQL standard part 2: Foundation defines a trigger as an action or multiple actions taking place as a result of an operation being performed on a certain object. These operations are known as trigger events, and are defined as changes made to rows by inserting, updating or deleting them. Therefore, three trigger types are being defined: the insert trigger, the delete trigger and the update trigger.

The timing of the triggered actions are also defined in the standard. An action can take place immediately before the operation, instead of the operation or immediately after the operation. A trigger is thus additionally defined as a BEFORE trigger, an INSTEAD OF trigger or an AFTER trigger. A BEFORE trigger would perform its action first and only then the operation that activated it would be performed. In the case of the AFTER trigger, the activation operation would be performed first, and then the trigger action would follow. In both cases the trigger action is executed in addition to the activation operation. The INSTEAD OF is different because the activation operation is replaced by the trigger action.

Another trigger property that is also defined in the standard, is how the triggered ac-

tion is to be applied to the object. Two cases are defined: the action can take place only once, or it can occur for every row in the object that the operation manipulates. The trigger is thus further defined as a statement-level trigger or as a row-level trigger. A complete trigger specification thus needs to indicate the trigger event (insert/update/delete), the trigger action time (BEFORE/INSTEAD OF/AFTER) and the application level (statement-level/row-level).

5.3 Standard Triggers

The first aspect that was looked at, was the conformance to the ISO/IEC 9075 SQL standard regarding the type of triggers. It was found that all surveyed DBMSs implement the three types of DML triggers defined in the standard. However, the only implementations that match the specification exactly with regards to trigger types, are those of Oracle and PostgreSQL. These DBMSs have implemented all combinations of BEFORE/AFTER/INSTEAD OF/Statement-level/Row-level triggers. The implementations of the other DBMSs either restrict the available combinations or implement only a subset of the definition from the specification.

Specifically, DB2 has no BEFORE statement-level trigger, but all the other combinations are implemented. Furthermore, SQL Server and Sybase do not implement BEFORE triggers at all. Also, they both do not have row-level triggers and all triggers are per default statement-level triggers. On the other hand, Mysql and SQLite do not have any statement-level triggers and all triggers are per default row-level triggers. This fact should be kept in mind when designing trigger actions since row-level triggers will have a significant overhead in tables with many rows of data.

PostgreSQL goes one step further and differentiates between the DELETE and TRUNCATE operation. Because the standard only specifies the DELETE operation, most databases will not execute the DELETE triggers when a TRUNCATE operation is performed. Depending on the viewpoint, this can be advantageous or problematic. It allows for the quick clearing of data from a table without having to perform possibly time consuming trigger actions. However, if a DELETE trigger was placed on a table to clean up data in other tables first, a TRUNCATE operation on that table might fail due to ref-

erential integrity constraints. The linked tables will have to be truncated in the correct order to be successfully cleared. PostgreSQL allows additional TRUNCATE triggers to be placed on such linked tables, facilitating easy truncation of related tables.

```
CREATE TRIGGER [test_dml]
ON dbo.Benefit
AFTER DELETE
AS
BEGIN
IF (ORIGINAL_LOGIN()= 'sa' OR ORIGINAL_LOGIN()= 'HOMEVM\whauger')
    INSERT INTO dbo.Benefit VALUES('S15', 'MY DML BENEFIT');
END;
```

Figure 5.1: DML Trigger Example. SQL Server Syntax.

Figure 5.1 shows a plain example of a standard (DML) trigger as used in SQL Server. As can be deduced from the keywords "DELETE" and "AFTER", this is a delete trigger where the specified SQL statement takes place immediately after a delete operation is performed. This trigger has been placed on a table called *dbo.Benefit*.

The SQL statement it contains, checks the user id that is performing the delete operation and if it matches specific privileged user accounts, it will insert a new row into the same table. Specifically, this example trigger will insert a single row into the table *dbo.Benefit* when one or more rows are deleted from the table *dbo.Benefit* by the privileged users "sa" or "whauger".

Since all three types of DML triggers defined by the standard rely on changes of data taking place i.e. either the insertion of new data, or the changing or removal of existing data, the standard methods employed by the forensic examiner are not impacted. These methods are specifically chosen because they do not cause any changes and can be used to create proof that, in fact, no changes have occurred.

Some members of the development community forums have expressed the need for a select trigger (Oracle, 2006). A select trigger would be a trigger that fires when a select operation takes place on the object on which it is defined. None of the DBMSs surveyed implement such a select trigger. Microsoft, however, is working on such a trigger and its researchers have presented their work already (Fabbri, Ramamurthy, & Kaushik, 2013). Oracle, on the other hand, has created another construct that can be used to perform

one of the tasks that the developers want to perform with select triggers: manipulate SQL queries that are executed. The construct Oracle has created is called a group policy. It transparently applies the output from a user function to the SQL executed on the defined object for a certain user group. The function can be triggered by selecting, inserting, updating or deleting data. The good news for the forensic examiner is that these functions will not be invoked for users with system privileges. As long as the forensic examiner uses a database user with the highest privileges, the group policies will not interfere with his examination.

The existence of a select trigger would have greatly impacted on the standard methods used by the database forensic examiner. One of the methods used to gather data and metadata for analysis, is the execution of SQL select statements on system and user database objects, such as tables and views. This would have meant that an attacker could have used such a trigger to hide or, even worse, destroy data. A hacker could use select triggers to booby-tap his root kit. By placing select triggers on sensitive tables used by him, he could initiate the cleanup of incriminating data or even the complete removal of his root kit should somebody become curious about those tables and start investigating.

5.4 Non-standard Triggers

The second aspect that was investigated was the additional types of triggers that some DBMSs define. The main reason for the existence of such extra trigger types is to allow developers to build additional, and more specialised auditing and authentication functionality, than what is supplied by the DBMS. However, that is not the only application area and triggers can be used for a variety of other purposes. For example, instead of having an external application monitoring the state of certain elements of the database and performing an action once certain conditions become true, the database itself can initiate these actions.

The non-standard triggers can be categorised into two groups: DDL triggers and other non-data triggers. From the DBMSs investigated, only Oracle, SQL Server and Sybase provide non-standard triggers.

5.4.1 DDL Triggers

The first group of non-standard triggers are the DDL triggers. These are triggers that fire on changes made to the data dictionary with DDL SQL statements (e.g. create, drop, alter). Different DBMSs define different DDL SQL statements that can trigger actions. SQL Server has a short list that contains just the basic DDL SQL statements. Oracle has a more extensive list and also a special DDL indicator that refers to all of them combined. Since DDL SQL statements can be applied to different types of objects in the data dictionary, these triggers are no longer defined on specific objects. Rather, they are defined on a global level, firing on any occurrence of the event irrespective of the object being changed. Both SQL Server and Oracle allow the scope to be set to a specific schema or the whole database.

```
CREATE TRIGGER [test_ddl]
ON DATABASE
AFTER DROP_TABLE
AS
BEGIN
IF (ORIGINAL_LOGIN()= 'sa' OR ORIGINAL_LOGIN()= 'HOMEVM\whauger')
    INSERT INTO dbo.Benefit VALUES('S16', 'MY DDL BENEFIT');
END;
```

Figure 5.2: DDL Trigger Example. SQL Server Syntax.

Figure 5.2 shows a plain example of a DDL trigger as used in SQL Server. The special reserved word "DROP_TABLE" indicates that this is a trigger that fires when tables are dropped. As can be deduced from the keyword "AFTER", this is a drop trigger where the specified SQL statement takes place immediately after the drop operation. Instead of being placed on a specific database object, this trigger has been associated with the entire database.

The SQL statement it contains, once again checks the user id that is performing the drop table operation and if it matches specific privileged user accounts, it will insert a new row into a table called *dbo.Benefit*. Specifically, this example trigger will insert a single row into the table *dbo.Benefit* when a table is dropped by privileged users "sa" or "whauger" in the database, where the trigger resides

As with DML triggers, such DDL triggers once again rely on data changes being made in the database to fire and thus they pose no problem of interference during the forensic examination.

5.4.2 Non-data Triggers

The second group of non-standard triggers are non-data triggers. These are triggers that fire on events that occur during the normal running and usage of a database. Since these triggers do not need any data changes to fire, they potentially have the biggest impact on the methods employed by the forensic examiner. Fortunately, the impact is isolated because only some DBMSs have implemented such triggers.

SQL Server, Oracle and Sybase define a login trigger. This trigger fires when a user logs into the database. SQL Server's login trigger can be defined to perform an action either before or after the login. Authentication, however, will be performed first in both cases, meaning only authenticated users can activate the trigger. That means the login trigger can be used to perform conditional login or even completely block all logins. An attacker could use this trigger to easily perform a denial of service (DoS) attack. Many applications now use some type of database connection pool that dynamically grows or shrinks depending on the load of the application. Installing a trigger that prevents further logins to the database would cripple the application during high load. It would be worse after an idle period where the application would have reduced its connections to the minimum pool size.

Oracle's login trigger only performs its action after successful login. Unfortunately, that distinction does not make a significant difference and this trigger can also be used to perform conditional login or completely prevent any login. That is because the content of the trigger is executed in the same transaction as the triggering action (Oracle, 2009a). Should any error occur in either the triggering action or the trigger itself, then the whole transaction will be rolled back. Simply raising an explicit error in the login trigger will reverse the successful login.

Sybase distinguishes between two different types of login triggers. The first is the login-specific login trigger. The trigger action is directly linked to a specific user account. This type of trigger is analogous to the facility some operating systems provide, which

can execute tasks on login. The second type is the global login trigger. Here, the trigger action will be performed for all valid user accounts. Sybase allows both types of login triggers to be present simultaneously. In this case, the global login trigger is executed first and then the login-specific trigger (Verschoor, 2007).

Both types of login triggers are not created with the standard Sybase SQL trigger syntax. Instead a two-step process is used. First, a normal stored procedure is created, that contains the intended action of the trigger. Then this stored procedure is either linked to an individual user account or made applicable to all user accounts with built-in system procedures. Like with Oracle, the action procedure is executed after successful login, but within the same transaction. Thus, it can be similarly misused to perform a DoS attack.

```
CREATE TRIGGER [test_login]
ON ALL SERVER
AFTER LOGON
AS
BEGIN
IF (ORIGINAL_LOGIN()= 'sa' OR ORIGINAL_LOGIN()= 'HOMEVM\whauger')
    INSERT INTO dbo.Benefit VALUES('S17', 'MY LOGIN BENEFIT');
END;
```

Figure 5.3: Login Trigger Example. SQL Server Syntax.

Figure 5.3 shows a plain example of a login trigger as used in SQL Server. The special reserved word "LOGON" indicates that this is a trigger that fires when users login. Together with the keyword "AFTER", it can be deduced that this is a login trigger where the specified SQL statement takes place immediately after successful login. Instead of being placed on a specific database or object inside this specific database, this trigger has been placed on the entire database server.

Once again, the SQL statement it contains, checks the user id that is logging into the database server and if it matches specific privileged user accounts, it will insert a new row into a table called *dbo.Benefit*. Specifically, this example trigger will insert a single row into the table *dbo.Benefit* when privileged users "sa" or "whauger" log into the database server.

Microsoft has considered the possibility of complete account lockout and subsequently

created a special method to login to a database that bypasses all triggers. Oracle, on the other hand, has made the complete transaction rollback not applicable to user accounts with system privileges or the owners of the schemas to prevent a complete lockout. Additionally, both SQL Server and Oracle have a special type of single-user mode the database can be put into, which will also disable all triggers (Microsoft, 2012b; Oracle, 2009b). Sybase, on the other hand, has no easy workaround and the database needs to be started with a special flag to disable global login triggers (Verschoor, 2007).

A hacker could use this trigger to check if a user with system privileges, that has the ability to look past the root kits attempts to hide itself, has logged in. Should such a user log in, he can remove the root kit almost completely, making everything appear normal to the user, even on deeper inspection. He can then use Oracle's BEFORE LOGOFF trigger to re-insert the root kit, or use a scheduled task (Kornbrust, 2005) that the root kit hides, to re-insert itself after the user with system privileges has logged off.

Another non-data trigger defined by Oracle is the server error trigger. This trigger fires when non-critical server errors occur and could be used to send notifications or perform actions that attempt to solve the indicated error.

The final non-data triggers defined by Oracle only have a database scope due to their nature: the database role change trigger, the database startup trigger and the database shutdown trigger. The role change trigger refers to Oracle's proprietary Data Guard product that provides high availability by using multiple database nodes. This trigger could be used to send notifications or to perform configuration changes relating to the node failure and subsequent switch over.

The database startup trigger fires when the database is opened after successfully starting up. This trigger could be used to perform certain initialisation tasks that do not persist, and subsequently do not survive a database restart. The database shutdown trigger fires before the database is shut down and could be used to perform cleanup tasks before shutting down. These last two triggers can be similarly exploited as the login and logoff triggers by a hacker to manage and protect his root kit.

5.5 Trigger Objects

The third aspect that was investigated, was which database objects the DBMSs allowed to have database triggers. The standard generically defines that triggers should operate on objects, but implies that the objects have rows. It was found that all DBMSs allow triggers to be applied to database tables. Additionally, most DBMSs allow triggers to be applied to database views with certain varying restrictions. Only Mysql restricts triggers to be applied to tables only.

None of the DBMSs allow triggers to be applied to system tables and views. Triggers are strictly available only on user tables and views. Additionally, there are restrictions to the type of user table and user views that triggers can be applied to.

This is good news for forensic examiners, since they are very interested in the internal objects that form part of the data dictionary. However, there is a move by some DBMSs to provide system procedures and views to display the data from the internal tables (Lee & Bieker, 2009). To protect these views and procedures from possible user changes they have been made part of the data dictionary. The ultimate goal appears to be to completely remove direct access to internal tables of the data dictionary.

This might be unsettling news for forensic examiners as they prefer to access any data as directly as possible to ensure the integrity of the data. It will then become important to not only use a clean DBMS, but also a clean data dictionary (at least the system parts). Alternatively, the forensic examiner first needs to show that the data dictionary is not compromised by comparing it to a known clean copy (Olivier, 2009). Only then can he use the functions and procedures provided by the data dictionary.

5.6 Conclusion

To determine if triggers could possibly interfere with a forensic examination by contaminating data, a more in-depth investigation of triggers was performed. First, the ISO/IEC 9075 SQL standard was examined on a theoretical basis. However, it was determined that various DBMSs manufacturers do not follow the standard exactly for different reasons. Thus, the actual trigger implementations of a number of prominent DBMSs were investigated.

This chapter established that there was indeed cause for concern. It was found that certain of the investigated DBMSs implemented additional triggers that were not specified in the standard. Of particular interest are the group of non-data triggers. One of the goals that the forensic examiner has, is to keep the data he is examining unchanged. Thus, triggers that rely on data changes to activate, are not a contamination risk. On the other hand, triggers that activate due to other actions, such as database startup or login, can be a risk.

This is because these actions might be performed by a forensic examiner during the cause of his examination. How exactly these non-data triggers can interfere with the forensic acquisition performed by the forensic examiner, is explored in the next chapter.

Chapter 6

Forensic Examination of Relational Databases

This chapter takes a closer look at the forensic examination of relational databases. Specifically, the implications of the presence of triggers during a forensic examination of a database are analysed. The previous chapter already indicated that certain types of triggers can potentially interfere with the acquisition process of a forensic examination. This chapter also looks at how the presence of triggers could interfere with the forensic interpretation process.

An analysis of a sample forensic examination on a relational database is conducted in section 6.1 in order to explore the potential interference of non-data triggers. Section 6.2 investigates how the presence of triggers, including standard triggers, can interfere with forensic attribution performed during the interpretation process. Section 6.3 then discusses the results further. Section 6.4 concludes this chapter.

6.1 Forensic Acquisition and Analysis Implications

In order to make the implications of the presence of non-data triggers more concrete, a look at some real-world forensic database examinations would be beneficial. Unfortunately, there is little scientific material available on such examinations. However, there is one examination that has been cited in a number of scientific papers: The forensic

examination of a SQL Server 2005 database performed in 2007 by Kevvie Fowler (2007b) for one of his clients. He provides a full account that lists in detail the exact steps that he followed during his examination.

Even though this detailed examination of a SQL Server database might be the only available scientific reference, certain fundamental activities carried out in this examination will be performed in all database examinations. These activities include accessing the database in order to extract information and data for preservation and analysis. Thus, it is worth testing what implications the presence of triggers has on the acquisition and analysis processes performed in this examination.

Of particular interest is the question if these processes are able to deal with the presence of triggers adequately. In order to do that, one needs to go through the examination one step at a time and question each step with regards to database triggers. Taking such a systematic approach will ensure that all areas where triggers could possibly interfere, are identified. The next section follows this step by step approach, up to a certain point, in the examination account of Fowler.

6.1.1 SQL Server Examination

In this examination, Fowler related how he was called out to a client company to investigate a possible security incident after a suspicious transaction was flagged by the credit card company of a client. The client company was running an online sales application that used a Microsoft SQL Server 2005 database. Since the system was mission critical for the client company, they did not want to take the application down unless a compromise could be detected. Fowler thus needed to perform a live analysis, acquiring and viewing volatile and non-volatile data using the live operating system directly on the database server

After having been logged into the console of the database server, Fowler inserted his Forensic Response CD, which contained trusted binaries of all the tools and utilities he might need. He used his own copy of the "net" utility to mount a remote shared drive from his forensic workstation that was connected to the network and accessible from the database server. He used this mapped drive to store all the information he gathered, including any forensic images he made.

The first tool that Fowler ran was the Windows Forensic Toolchest (WFT), which he used to gather volatile database and operating system information. WFT is a forensically enhanced batch processing shell that is capable of running other security tools and utilities. It provides a simplified way of scripting data collection activities, while computing MD5 or SHA1 checksums and logging all activities in an auditable fashion (Fool Moon, 2014). Since this output produced by WFT is forensically sound, it can be used in legal proceedings.

Fowler has provided the portion of his WFT configuration file that he added for SQL Server. A closer look shows that he scripted five SQL queries, which provide the database and system information he wants. He executes those queries with a command line utility called "sqlcmd" that is provided by Microsoft. He uses two command line options: one to specify the query and another to use a trusted connection. What a trusted connection means, is that instead of using provided or configured username/password credentials, the utility will use the logged in user account details to access the database server. In Fowler's case, this was the local administrator account.

This first step that Fowler performed, provides the exact scenario that was described in chapter 5: A privileged user logs into a potentially compromised database. A login trigger placed with malicious intent can now execute and interfere in various ways with the examination that Fowler was performing. It is therefore necessary to take precautions, in regard to triggers, before this first step is performed.

There are two obvious approaches one can take: Either, one needs the ability to assess the triggers in a database before one logs into the database, or one needs to access the database in such a manner that triggers are disabled or bypassed. However, triggers are just another database element that is stored and maintained inside the database. Providing direct access to them via some alternate mechanism, possibly without the same authentication and authorisation as used in the standard mechanism, would compromise the whole security model of the database.

The other option, to disable or bypass triggers, could also be seen as a possible security compromise. However, there are practical considerations that might force the hands of manufacturers of databases that support triggers to provide such an option. As already discussed in chapter 5, a login trigger could be used to perform a DoS attack on

the database. This could be either intentional or simply due to a mistake. The question is how one would regain control of a database when this has happened.

In the case of SQL Server, Microsoft has recognised this problem and consequently provided an option to bypass login triggers. Their option is a dedicated administrator connection (DAC) that runs on a separate TCP port (Microsoft, 2015). This connection is for diagnostic purposes with limited functionality, which can be used when the standard connection options no longer work. When connecting to the database using the DAC, no login triggers are executed; they are temporarily disabled (Microsoft, 2012a).

This DAC is accessed by simply specifying a specific command line option to the different Microsoft database clients and tools. In Fowler's case, the solution would have been as easy as adding another command line option to "sqlcmd" in the WFT configuration file. This would then allow Fowler to access the database without any login triggers running, thus mitigating the risk of a possible malicious trigger.

However, this additional command line option would only get Fowler into the database without setting off any login triggers. He would still need to check, if there are any login triggers present and active in the database. This could be done by adding another SQL query to the WFT configuration file, which lists all login triggers and their status.

After the WFT scripts had completed running, Fowler analysed the output that was produced. As part of this analysis, he could now also check for the presence of login triggers. If there are active login triggers in the database, he has two choices as to how to continue his examination safely. First, he could continue to use the DAC for all further database interaction. Alternatively, he could disable all the login triggers while connected via the DAC and then continue to use the standard connection for all further database interaction.

After having confirmed that there was indeed unauthorised access to the database, Fowler then continued by obtaining database configuration information. He used the same command line utility "sqlcmd" as in the WFT scripts. Different command line options were used that time to allow Fowler to log all output generated during his session to his forensic workstation.

What precautions Fowler would need to take regarding login triggers depends on what choice he made in the previous step. If he did disable any active login triggers, he

can continue using "sqlcmd" as he normally would. If, however, he decided not to make any changes to the database, he would need to add the same command line option to "sqlcmd" as was required in the WFT configuration file.

Once Fowler had completed acquiring the volatile data and configuration information, he shut down the database. Then, he continued acquiring the non-volatile data located in various files on the file system. He imaged them in a forensically sound manner using the "dcfldd" tool.

This tool was developed by Nicholas Harbour in 2002 while working for the Department of Defense Computer Forensics Lab (DCFL). "dcfldd" is an enhanced version of GNU "dd" file copy utility with features useful for forensics and security. These features include on-the-fly hashing of the input data being transferred, as well as the logging of all its output to files (Harbour, 2002).

Fowler then moved to his forensic workstation to start the analysis of the acquired data. He used various tools, including Microsoft Excel, to view the various captured and acquired files. However, to view the raw database pages, he attached the acquired database files to his SQL server. He then logged on to this database and used built-in tools to show raw data pages of interest.

Before Fowler logs into the database on his forensic workstation, he needs to take the same precautions as before. If he did disable any active login triggers, he can continue logging into the database as he normally would. If, however, he decided not to make any changes to the database, he would need to connect to the database using the DAC first. If the built-in tools are available via the DAC, he can continue using it. Otherwise he would have to examine any active login triggers present and disable those that make changes. Then he could logout and login again as he normally would.

Fowler has now completed the acquisition process and is busy with the analysis process. Since this example database examination is performed on a SQL Server database, the login trigger is the only non-data trigger available that could interfere. The areas in the examination process where login triggers could potentially interfere, have already been identified and discussed.

Continuing the step by step analysis at this point will provide no further insights. Had this been an examination of an Oracle database, then it would have been prudent to

continue analysing the example database examination in order to discover further areas of potential interference by some of the other non-data triggers that Oracle provides.

6.2 Forensic Interpretation Implications

The previous section concentrated on the potential impact that the presence of non-data triggers has on forensic acquisition and analysis during a forensic examination. This section moves the focus to forensic interpretation, specifically forensic attribution.

The concern of this section revolves around the automatic nature of actions performed by triggers. As an example, consider a database table that contains an AFTER update trigger that updates a value in the same table. This table with the trigger is then updated by a database user. Can the current attribution process correctly identify which party is responsible for which changes? This section attempts to establish if these concerns around triggers are justified.

6.2.1 Identification and Forensic Attribution

The login trigger problem identified in section 6.1.1 raises another interesting problem. Once the forensic examiner has pieced together all the actions, which occurred at the time when the user with system privileges was logged in, the attribution of those actions can then be performed. Since the forensic examiner can now make the assumption that the picture of the events that took place is complete, he would attribute all the actions to this specific user. This is because all the individual actions can be traced to this user account by the audit information. Without looking at triggers, the examiner will miss, that the particular user might have been completely unaware of certain actions that happened, even though they were activated and executed with his credentials.

The actions that are performed by triggers can be categorised into two groups: commission actions and omission actions. The trigger action time discussed in section 5.1 determines in which group a particular trigger action can fall. The BEFORE/AFTER trigger can be used to commission additional actions before or after the intended operation is performed. Since the original operation is still performed unchanged, no omission actions can be performed. The outcome of the original operation can still be changed or

completely reversed by actions performed in an AFTER trigger, but those actions are still commission actions.

The INSTEAD OF trigger, on the other hand, can be used to perform actions in both groups. Normally, this trigger is intended to commission alternative actions to the original operation requested. Similar to the BEFORE/AFTER trigger, it can also be used to commission actions in addition to the original operation. But importantly, it provides the ability to modify the original operation and its values. This ability also makes it possible to either remove some values or remove the operation completely. Operations that were requested, simply never happen and values that were meant to be used or stored, disappear. These removal actions therefore fall into the omission action group.

Consider a database in a medical system that contains patient medical information. An additional information table is used to store optional information, such as organ donor consent and allergies, in nullable columns. This system is used, among other things, to capture the information of new patients being admitted to a hospital. The admissions clerk carefully enters all the information from a form that is completed by the patient or his admitting physician. The form of a specific patient clearly indicates that he is allergic to penicillin. This information is dutifully entered into the system by the admissions clerk.

However, an attacker has placed an INSTEAD OF trigger on the additional information table that changes the allergy value to null before executing the original insert. After admission, the medical system is then used to print the patient's chart. A medical doctor then orders the administration of penicillin as part of a treatment plan after consulting the chart, which indicates no allergies. This action ultimately leads to the death of the patient due to an allergic reaction. An investigation is performed to determine the liability of the hospital after the cause of death has been established. The investigation finds that the allergy was disclosed on the admissions form, but not entered into the medical system. The admissions clerk that entered the information of the patient that died is determined and questioned. The admissions clerk, however, insists that he did enter the allergy information on the form and the system indicated that the entry was successful. However, without any proof substantiating this, the admissions clerk will be

found negligent.

Depending on the logging performed by the particular database, there might be no record in the database that can prove that the admissions clerk was not negligent. The application used to capture the information might, however, contain a log that shows a disparity between the data captured and the data stored. Without such a log, there will possibly be only evidence to the contrary, implying gross negligence on the part of the admissions clerk. This could ultimately lead to the admissions clerk being charged with having performed an act of omission. However, should triggers be examined as part of a forensic examination, they could provide a different perspective. In this example, the presence of the trigger can, as a minimum, cast doubts on the evidence and possibly provide actual evidence to confirm the version of events, as related by the admissions clerk.

The next example shows commission actions by using a trigger to implement the salami attack technique. An insurance company pays its brokers commission for each active policy they have sold. The commission amount is calculated according to some formula and the result stored in a commission table with five decimal precision. At the end of the month, a payment process adds all the individual commission amounts together per broker and stores the total amount, rounded to two decimals, in a payment table. The data from the payment table is then used to create payment instructions for the bank.

Now, an attacker could add an INSTEAD OF trigger on the insert/update/delete operations of the commission table, which would get executed instead of the insert/update/delete operation that was requested. In the trigger, the attacker could truncate the commission amount to two digits, write the truncated portion into the payment table against a dormant broker and the two decimal truncated amount, together with the other original values, into the commission table. The banking details of the dormant broker would be changed to an account the attacker controlled and the contact information removed or changed to something invalid so that the real broker would not receive any notification of the payment.

When the forensic examiner gets called in after the fraudulent bank instruction gets discovered, he will find either of two scenarios: The company has an application that

uses database user accounts for authentication or an application that has its own built-in authentication mechanism and uses a single database account for all database connections. In the first case, he will discover from the audit logs that possibly all users that have access in the application to manage broker commissions, have at some point updated the fraudulent bank instruction. Surely not all employees are working together to defraud the company. In the second case, the audit logs will attribute all updates to the fraudulent bank instruction to the single account the application uses.

In both cases, it would now be worthwhile to query the data dictionary for any triggers that have content that directly or indirectly refers to the payment table. Both Oracle and SQL Server have audit tables that log trigger events. If the trigger events correlate with the updates of the payment table as indicated in the log files, the examiner will have proof that the trigger in fact performed the fraudulent payment instruction updates. He can then move on to determine when, and by whom, the trigger was created. Should no trigger be found, the examiner can move on to examining the application and its interaction with the database.

Another more prevalent crime that gets a lot of media attention is the stealing of banking details of customers of large companies (Osborne, 2014). The most frequent approach is the breach of the IT infrastructure of the company and the large scale download of customer information, including banking details. This normally takes place as a single big operation that gets discovered quickly afterwards. A more stealthy approach would be the continuous leaking of small amounts of customer information over a long period of time.

Triggers could quite easily be used, to achieve the continues leaking of customer information at the insurance company in our previous example. The attacker can add an AFTER trigger on the insert/update operations of the banking details table. The trigger takes the new or updated banking information and writes it to another table. There might already be such a trigger on the banking details table for auditing purposes and so the attacker simply has to add his part. To prevent any object count auditing from picking up his activities, the attacker can use an existing unused table. There is a reasonable chance he will find such a table, because there are always features of the application that the database was designed to have, that simply were not implemented

and might never be. This is due to the nature of the dynamic business environment the companies operate in.

Suppose, every evening a scheduled task runs that takes all the information stored in the table, puts it in an email and clears the table. There is a possibility that some form of email notification method has already been setup for the database administrator's own auditing process. The attacker simply needs to piggy back on this process and as long as he maintains the same conventions, it will not stand out from the other audit processes. Otherwise, he can invoke operating system commands from the trigger to transmit the information to the outside. He can connect directly to a server on the Internet and upload the information if the database server has Internet connectivity. Otherwise, he can use the email infrastructure of the company to email the information to a mailbox he controls.

The forensic examiner that investigates this data theft will find the same two scenarios as in the previous example. The audit information will point to either of the following: All the staff members are stealing the banking information together or somebody is using the business application to steal the banking details with a malicious piece of functionality. Only by investigating triggers and any interaction with the table that contains the banking information, will he be able to identify the correct party responsible for the data leak.

The actual breach of the IT infrastructure and the subsequent manipulation of the database could have happened weeks or months ago. This creates a problem for the forensic examiner that tries to establish who compromised the database. Some log files he would normally use, might no longer be available on the system because they have been moved according to archive polices. If the compromise was very far back, some archives might also no longer be available because, for example, the backup tapes have already been rotated through and reused. It is very useful to the forensic examiner that a trigger was used in this example. The creation date and time of a trigger can give him a possible beginning for the timeline, and more importantly, the time window in which the IT infrastructure breach occurred. He can then use the log information that is still available for that time window to determine who is responsible for the data theft.

6.3 Discussion of Implications

This section summarises the findings made so far in this dissertation and then starts working towards addressing the issues raised.

Finding 1:

Chapter 5 identified a group of triggers classified as non-data triggers that do not require any data changes to activate. These triggers have the potential to contaminate the data in a database by inconspicuous operations such as connecting to it or shutting it down.

Section 6.1.1 showed that the way databases are acquired and analysed during a forensic examination, will make use of those operations. Thus, databases may be contaminated by the very person attempting to acquire and analyse the database without altering it. It does not matter whether the triggers in the database were placed intentionally or maliciously.

Finding 2:

In section 6.2.1, it was established that triggers can introduce side effects into the normal flow of operations. These side effects include performing additional actions or preventing the completion of the triggering operations. Certain types of triggers can also manipulate, or completely replace, the original operation. This means what the original operation intended to do, and what actually happened, are not necessarily the same.

As noted in that section, triggers can lead to incorrect conclusions when attributing operations performed in the database. This is because a trigger performs its actions with the same user credentials as the original operation that caused the trigger to fire. Some databases might log additional information with an operation to indicate that it was performed by a trigger. However, one cannot assume that such an extended log will be available to the forensic examiner.

In summary, the consequence of both findings is that the forensic examiner needs to be aware of triggers during an examination. Thus, the first step is to determine if there are actually any triggers present in the database under investigation. In order to do that, however, one needs to connect to the database. According to the first finding this can already lead to contamination of the data and thus precautions need to be taken.

A possible approach that one can take, was already given in section 6.1.1: one needs to access the database in such a manner that triggers are temporarily disabled or bypassed. Since there is no standard way of accessing a database in this manner, this special manner of accessing the database will be manufacturer specific and implementation dependent.

Once connected, all active non-data triggers can then be disabled properly. Thereafter, a full check for all types of database triggers can be performed. If triggers are indeed present, the next step is to establish if any of those triggers could have influenced the data that is to be analysed.

6.4 Conclusion

This chapter investigated the potential implications that the presence of triggers can have on forensic database examinations. First, the impact of non-data triggers on the acquisition and analysis process was explored in more detail. Then, the influence that the presence of triggers, including standard triggers, has on forensic attribution was investigated.

Based on the implications discovered in this chapter, it has become clear that database triggers have to be considered during the acquisition, analysis and interpretation phases of a forensic examination. A database under investigation can potentially contain hundreds or even thousands of triggers. It is thus not feasible for the forensic examiner to analyse every single one to determine if any of them had an impact on the data that is being examined.

A more efficient way is required to identify any potential triggers that need to be analysed as part of the examination. It would also be valuable for a forensic examiner to have a standard method to test for the presence of triggers before commencing a forensic examination of a database. A method to test for triggers, and identify the ones that are potentially relevant, is devised and presented in the following chapter.

Chapter 7

Trigger Identification Algorithm

This chapter introduces a systematic approach to identify triggers that are potentially significant to the examination of a specific database object. The intention is to formally define a general algorithm that is database independent and practical to implement. This chapter is based on work from a previously published paper (Hauger & Olivier, 2015a).

Section 7.1 explores a general procedure that can be followed to determine the possible influence of a trigger on a specific database object. Thereafter, section 7.2 explores the steps of an algorithm that follows this approach and provides a formal definition. The implementability of the steps in a programming language is also assessed. Section 7.3 introduces another less intuitive algorithm that is based on the previous algorithm. This algorithm takes a different approach that provides the same outcome as the original algorithm, but promises better implementability. A formal definition of the new algorithm is also provided. Section 7.4 concludes this chapter.

7.1 Trigger Identification Considerations

A database stores a collection of data that may be related. In a relational database, this data is stored in tables. A DBMS allows the data to be accessed and manipulated. When this database is part of an active system, the data in the database will be constantly changing. The purpose of the system and the way the database has been employed will

determine how the data is changing.

In an archival system, for example, data will be mainly added and accessed. In a transactional system, on the other hand, data will also be updated and deleted. When the data in a specific database object is forensically examined to determine the events that led to the current state of the data, an important question needs to be asked. Could a database trigger have influenced the creation or change of a specific object, or a value that the object contains?

An easy way to answer this question would be to check if the name of that object was mentioned in any trigger. If such triggers are identified, the next step would be to manually check what type of SQL statement referred to the database object in question. If the SQL statement in question only read data, such as a SELECT statement, then there would be no impact on the data in the target object. Should the SQL statement, however, write or modify data using a statement such as INSERT, UPDATE or DELETE, then there is an impact on the data if the specific database object is the target of this SQL statement.

However, a trigger can not only change data in database objects directly. The trigger could also call user functions and procedures that contain similar SQL statements that refer to the same database object. Therefore, it is not enough to search just the trigger content since the SQL statements that refer to the specific database object could be encapsulated in a called user function or procedure.

The content of any called user function or procedure also needs to be checked in the same way. Since more user functions and procedures can be called inside a user function or procedure, the content of all those called also needs to be checked. This process needs to be repeated until the lowest level user function or procedure has been reached that calls no further user function or procedure.

The SQL standard classifies both SQL functions and SQL procedures as SQL invoked routines (ISO/IEC JTC1/SC32, 2011). Both are considered to be types of SQL invoked routines and there are only some defined differences between them. Firstly, functions and procedures are called differently to encourage different use cases for each. Procedures are invoked on their own with the keyword CALL. Functions, on the other hand, can also be called inside a SQL statement. Secondly, functions always return something, while

the parameter definition of a procedure determines whether it returns something or not. Thirdly, parameters defined for a function are always input parameters. The RETURNS keyword defines the type of the returned data, which can be either a value or a table. In the case of a procedure, parameters can be defined as either input or output parameters, or both.

Even though the syntax for SQL routines is standardised, many databases implement non-standard versions of this syntax (Silberschatz et al., 2011). SQL Server, as shown in figure 7.1, uses the keyword EXECUTE or EXEC to call procedures. In certain circumstances, procedures can even be called without any keyword.

```
-- Procedure
EXECUTE dbo.test_procedure;
GO
-- Or
EXEC dbo.test_procedure;
GO
-- Or, if this procedure is the first statement
dbo.test_procedure;

-- Procedure with parameter
EXECUTE dbo.test_proc 'S1';
GO
-- Or
EXEC dbo.test_proc 'S1';
GO
-- Or, if this procedure is the first statement
dbo.test_proc 'S1';
```

Figure 7.1: Procedure Execution. SQL Server.

One of the reasons for this is that these databases implemented user functions and procedures before they were standardised. To maintain backward compatibility with their original implementations, they now continue to support their own versions of the syntax. These versions can place additional or different restrictions on user functions and procedures, which leads to differences between functions and procedures that are database specific.

To simplify the formal definition of trigger identification algorithms, such as the one described above, the term *routines* as used in the SQL standard, which includes both

procedures and functions, will be used going forward.

7.2 Top-Down Approach

The procedure that is described in the previous section starts with a pool that contains all the triggers found in the database. The content of each trigger is searched for the name of the specific database object. If the object name is found, the trigger is then placed in another pool that requires further SQL statement analysis. If the object name is not found, the trigger is searched again for all routines that it calls. The content of each routine found, is then also searched for the name of the database object. If the object name is found, the trigger that called the routine is also placed into the SQL analysis pool. If the object name is not found, the routine is searched again for all routines that it calls in turn. This process is repeated until the triggers or routines call no further routines.

In this way, an exhaustive search is performed that drills down in each trigger, through the used routines, to the bottom routines. A search tree is created in the process, where the root at the top represents the trigger. The nodes below the root represent all the called routines inside the trigger content and the leaves represent the lowest routines, which call no further routines. Due to the possible structure of the created tree, this algorithm can be considered a top-down algorithm.

This top-down procedure that searches the content of triggers has been formalised in algorithm variation 1.

Algorithm Variation 1 (Top-down): Identify all triggers that access directly or indirectly a specific database object.

1. Assume triggers are ordered pairs (t, b_t) where t is the name of the trigger and b_t is the code (or body) of t. Assume further that routines exist and have a similar form (f, b_f) where f is the routine name and b_f is the body of f. The body b_x of a trigger or routine x consists of primitive statements as well as calls to other routines. Let $f \in b_x$ indicate that f is used in the body of trigger or routine x.

2. Let T be the set of all triggers and F be the set of all routines. Let ω be the name of the object being searched for. Let $\omega \in b_x$ indicate that ω is used in the body of

trigger or routine x. Let R_1 be all triggers that access ω directly. In other words

$$R_1 = \{t \in T \mid \omega \in b_t\} \tag{7.1}$$

3. Routines may be called by a trigger, where the routine accesses ω, thereby providing the trigger with indirect access to ω. Therefore, the $\in$ notation is extended to also indicate indirect access. The new relationship is denoted as $\in^*$. Then

$$f \in^* b_x \Leftrightarrow \left\{ \begin{array}{ll} f \in b_x & \text{direct case} \\ \exists\, b_y \ni f \in b_y \;\&\; y \in^* b_x & \text{indirect case} \end{array} \right\} \tag{7.2}$$

The complete set of routines used directly or indirectly by a trigger t is therefore

$$F^t = \{f \mid f \in^* b_t \ni t \in T\} \tag{7.3}$$

The subset of those routines that access ω is then simply

$$R_2 = \{t \in T \mid \exists\, f \in F^t \ni \omega \in b_f\} \tag{7.4}$$

The triggers that directly and indirectly access ω are then combined as

$$R = R_1 \cup R_2 \tag{7.5}$$

This algorithm will produce a set R of all triggers that refer directly or indirectly to the object of which the data is being analysed.

At first glance, this algorithm appears to be easy and straightforward to implement. However, when one translates the steps into a programming language (or query language such as SQL), one encounters a problem: How does one reliably identify other routines inside the trigger content?

The SQL standard specifies that procedures are always invoked with the CALL keyword. But that is only true for SQL procedures. However, as already highlighted in section 7.1, some databases use a different syntax from the standard. SQL Server, as shown in figure 7.1, uses the EXECUTE or EXEC keyword instead. In certain cases procedures can even be invoked without the keyword.

Regarding functions, the SQL standard specifies that SQL functions can also be used directly in other SQL statements. This makes functions even more difficult to identify.

```
-- Scalar function
declare @result int;
EXEC @result = dbo.test_funct;
select @result;
GO
-- Or
select dbo.test_funct() as result;

-- Table function with parameter
select * from dbo.test_func('S1');
```

Figure 7.2: Function Execution. SQL Server.

Function names are generally followed by "()" brackets which contain zero or more parameters. However, the SQL Server syntax, as shown in figure 7.2, does not require them in certain cases.

Therefore, simply searching for all strings that follow a specific keyword to identify procedures is not nearly enough. Similarly, to scan for the "(" bracket to identify functions is also not sufficient. As figures 7.1 and 7.2 show, routines can also be invoked in other ways. An additional complication is that the SQL syntax for invoking routines can also differ between databases.

Another approach to identifying routines could be to perform string matching of known strings in the trigger or routine content. One could retrieve the list of names, of all the user-defined SQL routines, and then search for each of those names in the content of the trigger.

However, this approach has its own drawbacks. Depending on the naming convention used by the developers of the database, the names of different object types could overlap. For example, a table, a view and a procedure could all have the exact same name. Should such an overlapping name appear in a trigger and not refer to a routine, the algorithm will unnecessarily evaluate the routine of that name.

The problem does not stop there. Should the unnecessarily evaluated routine directly or indirectly reference the object being examined, a false link will be made back to the trigger that initiated the evaluation. Therefore, this approach could produce false matches, leading to the incorrect identification of triggers.

7.3 Bottom-Up Approach

Due to the potential problems with the top-down approach, it is prudent to try and identify a different approach for finding the triggers that directly or indirectly refer to a database object being examined. Alternatively, one can begin with all triggers, functions and procedures and then narrow them down to only those, which refer to the object under investigation. Following this bottom-up approach, algorithm variation 2 is now formally defined.

Algorithm Variation 2 (Bottom-up): Identify all triggers that access directly or indirectly a specific object.

1. Assume triggers are ordered pairs (t, b_t) where t is the name of the trigger and b_t is the code (or body) of t. Assume further that routines exist and have a similar form (f, b_f) where f is the routine name and b_f is the body of f. The body b_x of a trigger or routine x consists of primitive statements as well as calls to other routines. Let $f \in b_x$ indicate that f is used in the body of trigger or routine x.

2. Let T be the set of all triggers and F be the set of all routines. Let ω be the name of the object being searched for. Let $\omega \in b_x$ indicate that ω is used in the body of trigger or routine x. Let C be the combined set of all triggers T and routines F. In other words

$$C = T \cup F \tag{7.6}$$

3. Let U be the set of all triggers and routines that access ω directly. That is

$$U = \{c \in C \mid \omega \in b_c\} \tag{7.7}$$

Let U^t be the subset of all the triggers in set U. Expressly

$$U^t = \{t \in U \mid t \in T\} \tag{7.8}$$

The subset U' of U without any triggers is then simply

$$U' = U - U^t \tag{7.9}$$

Let U_1 be the first iteration of a set of triggers and routines that access the routines in set U' directly. Specifically

$$U_1 = \{d \in C \mid \exists\, e \in b_d \ni e \in U'\} \tag{7.10}$$

Let U_1^t be the first subset of all the triggers in the first set U_1. Namely

$$U_1^t = \{t \in U_1 \mid t \in T\} \tag{7.11}$$

The first subset of U_1 without any triggers is then simply

$$U_1' = U_1 - U_1^t \tag{7.12}$$

Congruently sets U_2, U_2^t and U_2' can be constructed. Specifically, the construction of those sets can be repeated i times where $i = 1, 2, 3, ...$ Consequently, the i^{th} iteration of set U_1 is then

$$U_i = \{x \in C \mid \exists\, y \in b_x \ni y \in U_{i-1}'\} \tag{7.13}$$

Furthermore, the i^{th} subset of all the triggers in the set U_i is

$$U_i^t = \{t \in U_i \mid t \in T\} \tag{7.14}$$

Finally, the i^{th} subset of set U_i without any triggers is

$$U_i' = U_i - U_i^t \tag{7.15}$$

The combined set of all identified triggers $U^t, U_1^t ... U_i^t$ is then

$$R = U^t \cup U_1^t \cup ... \cup U_i^t \tag{7.16}$$

Since the set C is a finite set of triggers and routines, the algorithm will reach a point where set R no longer grows. Specifically, this is when the n^{th} set U_n becomes empty, in other words $U_n = \{\}$. At this point the algorithm can be terminated.

As can be seen, this variation of the algorithm also produces a set R of all triggers that refer directly or indirectly to the specified database object. However, it eliminates the need to identify calls to routines, which proved to be difficult and database specific. Rather, it repeatedly searches the content of routines and triggers for known routine names in the same way it searches for the specified object name.

The important difference to the previous algorithm variation is that only the names of routines that are known to reference the specified database object are being searched for. This will improve the performance of the algorithm and reduce the number of false positives.

7.4 Conclusion

This chapter proposed a systematic procedure to identify triggers that can potentially influence a specified database object. Two algorithm variations that followed different approaches were offered and formally defined. The implementability of both algorithm variations into a programming language was also assessed. It was found that the second algorithm variation would allow a more generic implementation that is also database independent. The first algorithm variation would require a database specific implementation that also had a greater potential of false matches.

The following chapter now investigates the implementation of the second variation of the algorithm. A prototype is built to evaluate the algorithm on a number of different databases using test cases. The results of the implementation and test cases are also analysed in this chapter.

Chapter 8

Algorithm Implementation

An algorithm to determine if triggers are present in a relational database was presented in the previous chapter. In order to verify this algorithm in practice, a prototype was built and used on a number of databases with certain test objects. The goal was to evaluate the usability of the prototype. During the implementation and usage of the prototype, a number of challenges were encountered. This chapter is based on work from a previously published paper (Hauger & Olivier, 2015a).

Section 8.1 explores the intended implementation of the proposed algorithm by looking at different implementation languages and options. Thereafter, section 8.2 presents the prototype that was subsequently built to verify the algorithm presented in chapter 7. Section 8.3 provides the prototype implementation details. Then section 8.4 discusses the challenges that were encountered with the implementation of the algorithm as well as the usage of the prototype. In section 8.5, the test setup is detailed. Section 8.6 concludes this chapter.

8.1 Implementation Considerations

The algorithm implementation is considered against the same databases that were discussed in chapter 5. In that chapter, it was shown that those chosen relational databases represented the biggest market share of used databases.

The databases considered are all relational databases, so it can be assumed that data

and metadata about triggers, functions and procedures are stored in relations. The most straightforward choice therefore, is to implement the algorithm using SQL. Each step of the algorithm can be written as a simple SQL statement. SQL statements that need to be repeated, can be placed into functions. All the separate SQL steps and functions can then be combined into a procedure.

One important aspect to keep in mind is that the database implementations available from various manufacturers vary considerably. This includes the design of the data dictionary. Which information is stored and how it is stored will differ between various databases. The retrieval of all triggers, for example, could be a single SELECT statement in one database implementation. In another database implementation, it might require performing multiple SELECT statements from multiple differently structured tables. Due to these considerable differences, it is not possible to create template SELECT statements that could be completed with the column and table names corresponding to a particular database. Instead, it is necessary to provide database-specific SQL statements for each database implementation.

A pure SQL implementation has two major drawbacks. The first drawback is that database manufacturers use different SQL dialects to extend the standard SQL statements. These extended features, such as variables, loops, functions and procedures, are required to implement the algorithm. Oracle uses an extension called PL/SQL which IBM's DB2 since version 9.7 now also supports (Chan, Ivanov, & Mueller, 2013; Feuerstein & Pribyl, 2014). Microsoft has a competing extension called Transact-SQL, or T-SQL in short, which SQL Server and Sybase use (Jones, 2005; Turley & Wood, 2009). Other databases, such as MySQL, more closely follow the official SQL/PSM extensions that are part of the SQL standard (ISO/IEC JTC1/SC32, 2008). These differences would require a separate implementation for every particular database product. The second drawback is that this implementation has to be stored and executed within the database being examined. However, this conflicts with the goal of the forensic examiner to keep the database uncontaminated.

Another approach is to implement the algorithm using a pure programming language. This provides the advantage of being able to create only one application using a single syntax. The application is designed to read the database data from a file that conforms

to a specific format. The drawback with this design is that the data first has to be extracted from the database being examined in the correct format. This requires a separate extraction procedure for every database implementation. The extraction procedure also may have to be stored and executed on the database, which potentially contaminates the database.

The latter implementation choice requires a more involved two-step process. First, the required data is extracted from the database and transformed into the format expected by the application by using an extraction procedure. Next, the standalone application is invoked with the created data file.

A combined approach that eliminates the drawbacks of both previous choices appears to be a better solution. A single application can be built using a conventional programming language that provides a database independent framework for accessing and using databases. Next, database-specific formatted SQL statements in the application are used to select and retrieve the data from the particular database being examined. The database would remain unchanged because the algorithm logic, as well as the SQL statements, are being stored and maintained external to the database.

8.2 Prototype Design

To verify the algorithm practically, a prototype that uses the presented combined approach was built. This prototype implements the bottom-up version of the algorithm that was presented in the previous chapter in section 7.3. The prototype consists of a stand-alone application that has been created in the Java programming language. It connects to the database that is being examined via the Java Database Connectivity (JDBC) application programming interface (API).

JDBC is the industry standard for database-independent connectivity between the Java programming language and a wide variety of SQL databases and other tabular data sources, such as spreadsheets and flat files (Oracle, 1998a). The JDBC API contains two different types of interfaces: the higher level API for application developers and the lower level API for database driver developers.

Database independence is achieved by the JDBC API as follows. The higher level

API provides a single, standardised set of database methods to the Java programming language. These methods allow the connection to a data source and the manipulation of the data that the data source contains.

A JDBC driver will then convert these method calls to a database or middleware specific protocol. The type and level of translation performed by the JDBC driver determines the type of JDBC driver. The lower level API ensures the same static interface for each JDBC driver. This allows different JDBC drivers to be used interchangeably. It enables, for example, the exchange of JDBC drivers of different types for the same database. Similarly, the change to a database from a different manufacturer is achieved by the substitution to a matching JDBC driver.

A JDBC driver can be classified in four different types : type 1 and 2 drivers make use of existing database client libraries, while type 3 and 4 drivers use pure Java implementations. Type 1 drivers use the Open Database Connectivity (ODBC) bridge to access the client libraries, while type 2 drivers interface directly with them. Type 3 drivers connect via database middleware to the database, while type 4 drivers connect directly to the database via native protocols (Oracle, 1998b).

The type of the JDBC driver can thus influence the integrity of the data returned by the database. Only a type 4 JDBC driver will connect directly via the native network protocol to the database in the same way a forensic examiner would connect to it (see section 6.1.1). The other JDBC driver types access the database via an intermediary: type 1 and 2 via client libraries and type 3 via middleware. Should a type 4 driver not be available for a particular database, untainted versions of the database client libraries or middleware will need to be used to ensure that the returned data is uncontaminated.

A suitable JDBC database driver is thus required by the prototype to handle the database communication for each particular database and its configuration. In order to use a selected JDBC driver, its driver file needs to be made available to the Java application execution environment and configured to use with the database under investigation. The configuration information can be placed either inside the code or in an external configuration file.

Once the database connection has been established, the required data for the algorithm is retrieved. This is achieved by executing database specific SQL SELECT

statements against the database being examined. This can be a simple SELECT statement from a single table, or multiple varying SELECT statements from the same or multiple tables. This depends on the complexity and layout of the database dictionary in question.

These SQL statements, together with the database and JDBC driver configuration, have been externalised to a definition file. This is realised using the Java object called "Properties". An instance of this object stores a list of key/value pairs. The list is populated from a specifically formatted text file called a properties file. The properties file stores information in key/value pairs. This setup allows these pieces to be updated or changed without the need to modify the application.

Consequently, the architecture of this prototype allows it to be extended to support any SQL database with triggers. This is achieved by simply defining the new database, adding the required database and JDBC driver configuration information, and providing the database specific SQL statements for each required part of the algorithm. The availability of a JDBC or ODBC driver, however, is a pre-requisite.

8.3 Prototype Implementation Details

The prototype was implemented using Java Standard Edition 7 (SE7). The two databases that were evaluated with the prototype are Microsoft SQL Server 2005 and Oracle Database 11g Release 2.

The JDBC database driver that was used for the SQL Server database is the jTDS v1.1 driver. jTDS is an open-source type 4 JDBC driver for Microsoft SQL Server and Sybase Adaptive Server Enterprise (ASE) (jTDS Project, 2018). For the Oracle database, the Oracle JDBC Thin client-side driver v10.2 was used. This is also a type 4 JDBC driver (Oracle, 2018).

SQL Server places triggers into three separate groups: DML, DDL and Non-data. These categories were already discussed in chapter 5. The DML and DDL triggers are tied to a specific database schema and stored in set of system views. The non-data triggers are global to all database schemas and are stored in a different set of system views.

Therefore, to retrieve a complete trigger list, two separate SQL retrieval queries are required. The DML and DDL triggers returned are also limited to only those that are in the same scope as the database connection used.

User functions and procedures are placed together, and can be retrieved from another set of system views. SQL Server does distinguish between three types of user functions: scalar functions that return single values, in-line table functions and normal table functions that both return result sets. The SQL retrieval query needs to cater for all types of user functions.

```
select t.object_id, t.name, 'T' as type_code,
te.type_desc as type, t.create_date, t.modify_date
from sys.server_triggers t, sys.server_trigger_events te
where t.object_id = te.object_id
and OBJECT_DEFINITION(t.object_id) like ?
UNION
select t.object_id, t.name, 'T' as type_code,
te.type_desc as type, t.create_date, t.modify_date
from sys.triggers t, sys.trigger_events te
where t.object_id = te.object_id
and OBJECT_DEFINITION(t.object_id) like ?
UNION
select m.object_id, ao.name,
case when ao.type = 'P' then 'P'
     when ao.type = 'FN' then 'F'
     when ao.type = 'TF' then 'F'
     when ao.type = 'IF' then 'F'
end as type_code,
ao.type_desc as type, ao.create_date, ao.modify_date
from sys.all_sql_modules m, sys.all_objects ao
where m.object_id = ao.object_id
and ao.type in ('P','FN','TF','IF')
and m.definition like ?
```

Figure 8.1: Algorithm Search Query. SQL Server.

Figure 8.1 shows the combined SQL query required for SQL Server to retrieve all triggers, functions and procedures. The "?" place-holder will be replaced dynamically by the Java code with the name of the database object being examined.

Oracle also differentiates between the same three trigger groups. However, instead of storing them in separate locations, they are all stored in the same system view. A field

in the view indicates what a trigger is tied to. DML triggers are tied to a specific table object, while DDL triggers are tied to a schema. Non-data triggers can be tied either to a single schema or the whole database.

As with SQL Server, the returned triggers are also limited to only those that are in the same scope as the database connection used. User functions and procedures are also placed together and can be retrieved from the same set of system views as the triggers. The content of all the database objects is handled differently by Oracle. While SQL Server stores all the content of a database object in a single field, Oracle stores each content line separately. This implies that when the name of the database object is found more than once in the content, the owner of the content will be returned multiple times. To eliminate these duplicates, the DISTINCT keyword is needed.

```
select distinct o.object_id, o.object_name,
case when o.object_type = 'TRIGGER' then 'T'
     when o.object_type = 'PROCEDURE' then 'P'
     when o.object_type = 'FUNCTION' then 'F'
end as type_code,
o.object_type as type , o.created, o.last_ddl_time
from sys.all_objects o, sys.all_source s
where o.object_name = s.name
and o.object_type in ('TRIGGER', 'PROCEDURE', 'FUNCTION')
and s.text like ?
```

Figure 8.2: Algorithm Search Query. Oracle.

Figure 8.2 shows the comparatively simple SQL query required for Oracle to retrieve all triggers, functions and procedures. As before, the "?" place-holder will be replaced dynamically by the Java code with the name of the database object being examined.

Figure 8.3 shows the Java code that contains the core of the bottom-up search algorithm. In line 372 the query either from figure 8.1 or 8.2 is loaded. The object name being searched for is set in line 374 and the query is then executed.

The results returned by the query are evaluated one by one. If the object returned is a trigger, it will be added to the final trigger list. All other returned objects have their name added to a temporary object list in line 388. A search-depth level is being maintained in line 392 so that the triggers added to the trigger list can be correctly

```
371         //Find object name
372         String dbQuery = objectContentMatchMap.get(dbms[dbmsNumber]);
373         stat = conn.prepareStatement(dbQuery);
374         stat.setString(1, name);
375         rs = stat.executeQuery();
376 
377         while (rs.next()) {
378             long objId = rs.getLong(1);
379             String objName = rs.getString(2);
380             String objType = rs.getString(3);
381             if (objType.equalsIgnoreCase("T")) {
382                 if (level == 1)
383                     dbTriggerList.add("Direct - " + objId + " " + objName +
384                 else
385                     dbTriggerList.add("Indirect - " + objId + " " + objName
386             }
387             else {
388                 dbObjectList.add(objName);
389             }
390         }
391         cleanup(null, stat, rs);
392         level++;
393 
394         //Call self
395         if (!dbObjectList.isEmpty()) {
396             for (String findname : dbObjectList) {
397                 doSearchObject(conn, findname);
398             }
399         }
```

Figure 8.3: Bottom-up Algorithm. Java Code.

labelled as direct or indirect references to the database object.

If the object list is not empty, it will now be iterated through. For each name in the list, the core of the search algorithm will be repeated. This is done using recursion in line 397.

8.4 Implementation Challenges

In this section, some of the challenges that were encountered with the implementation and usage of the proposed trigger identification algorithm are discussed. Some of the challenges are related to the general SQL syntax while others are related to database-specific implementation decisions.

8.4.1 Scope/Visibility

Most of the evaluated databases do not allow direct access to the data dictionary tables. These tables contain, among other things, the lists of all triggers, functions and procedures in the database. Instead, these databases provide access to the data in the tables via system views. The problem with these views is that they limit the results based on the permissions possessed by the user who accesses the view, and on the scope of the database connection.

Therefore, the user account that is being used to query the views needs to have access to all the objects. However, even if the user account has the permissions necessary to view all the objects, only those that fall within the scope of the connection will be listed. To work around that challenge, it is necessary to obtain a list of all the database scopes and iterate through them, changing the scope of the connection and repeating the query. The results of all the individual queries are then combined.

8.4.2 Encryption

Some of the evaluated databases have the functionality to encrypt the contents of database objects including triggers, functions and procedures. This makes it impossible to search the content for the name of the database object under investigation. Due to flaws in the current encryption implementations of the evaluated databases, it is possible to retrieve the decrypted content of an encrypted object (Cherry, 2012; Litchfield, 2007).

The prototype showed that these mechanisms can be used to work around this challenge. However, it was found that these workarounds are not very efficient and practical.

Furthermore, the current flaws that are being exploited might be corrected in a future version, making the implemented workaround worthless for those updated versions.

8.4.3 Case Sensitivity

Since SQL is mostly case insensitive, the name of the database object that is being searched can be spelled in many different ways. Therefore, searching the content for an exact match, based on a particular spelling, would be a hit and miss affair. The standard solution to overcome this challenge, is to lower case the content, as well as the name of the database object being looked for, prior to the comparison (Gulutzan & Pelzer, 2003). This is usually done using a built-in function.

```
select distinct o.object_id, o.object_name,
case when o.object_type = 'TRIGGER' then 'T'
     when o.object_type = 'PROCEDURE' then 'P'
     when o.object_type = 'FUNCTION' then 'F'
end as type_code,
o.object_type as type , o.created, o.last_ddl_time
from sys.all_objects o, sys.all_source s
where o.object_name = s.name
and o.object_type in ('TRIGGER', 'PROCEDURE', 'FUNCTION')
and lower(s.text) like lower(?)
```

Figure 8.4: Case Insensitive Search Query. Oracle.

Figure 8.4 shows how this solution can be applied to the Oracle search query shown earlier. For databases with not many objects that need to be searched, this solution is feasible. However, in a database with thousands of objects that could each have a content of thousands of lines, this could become very inefficient and impractical. In that case, a more efficient approach would be to change the collation of the database to enable case-insensitive comparison.

8.4.4 False-Positive Errors

Since basic string matching is performed, any string in the trigger, procedure or function would match the database object name that is being searched for. This could include

comments, variable names and the names of other object types.

To deal with this challenge, it will be necessary to manually inspect all the triggers listed by the algorithm. Any false-positives discovered can then be removed from the list. Since the SQL parser can identify comments based on the syntax, it is possible to pre-process the triggers, procedures and functions to remove the comments before initiating the search. However, that still leaves the other sources of matching strings that could produce false-positives.

8.4.5 Data Types

Not all of the evaluated databases store the content of triggers, functions and procedures in a table column with the same data type. Many databases have moved from using the simple VARCHAR data type to one that can hold more data. Microsoft SQL Server, for example, uses the Character Large Object (CLOB) data type to store content while Oracle uses an older LONG data type.

The challenge is that all the data types that are used, cannot be handled in the same way in the code. To prevent having to create different code paths for each encountered data type in order to accommodate these specific databases, a more generic approach is needed. The approach would entail querying the database table metadata first, to detect the data types that are used, and then invoke the relevant code to handle the specific data types.

8.4.6 Recursion

Database functions and procedures are allowed to call themselves either directly or indirectly. The level and depth of recursion allowed differs between various evaluated databases. Additionally, each database can be setup and configured differently for recursion. Thus, this challenge needs to be addressed at the algorithm level.

The possible recursion of functions and procedures would produce an endless loop in both variations of the algorithm. In the bottom-up algorithm, set R would still stop growing at some point because no new elements are added. However, set U_n would not become empty, which means the termination criteria are never met. To deal with this

problem, the algorithm needs to keep track of all elements seen in U_i' before. U_n' should not contain any elements evaluated before, specifically the following should hold true:

$$U_n' \cap U' \cup U_1' \cup ... \cup U_{n-1}' = \{\} \tag{8.1}$$

To achieve this, line (7.15) of the bottom-up algorithm can be modified as follows:

$$U_i' = U_i - U_i^t - \bigcup_{j=1}^{i-1} U_j \tag{8.2}$$

This change will ensure that each function and procedure is only evaluated once.

```
405  //Call self
406  if (!dbObjectList.isEmpty()) {
407      for (String findname : dbObjectList) {
408          if (!dbSeenList.contains(findname))
409          {
410              dbSeenList.add(findname);
411              doSearchObject(conn, findname);
412          }
413      }
414  }
```

Figure 8.5: Updated Algorithm Section. Java Code.

Figure 8.5 shows how this solution can be applied to the recursive section of the search algorithm shown earlier.

8.4.7 Performance

Maintaining adequate performance of the algorithm is another challenge. The string matching component of the bottom-up algorithm emerged as a potential bottleneck. In the previous chapter in section 7.3, the string matching component was defined as $\omega \in b_x$. This indicates that ω is used in the body of trigger or routine x.

This step can be performed with the SQL "LIKE" command. The object name ω being searched for can be located anywhere in the body of the trigger or function being searched. Thus, the SQL wild-card character "%" needs to be added around the object

name as in "%*object_name*%". This makes the execution of the SQL statement very slow since no index can be utilised.

In a database with not many procedures, functions and triggers, this does not pose a big problem. However, should there be many such objects, this SQL statement can take quite a while to complete. This may occur in systems where auditing is performed with triggers or where the application utilising the database might perform a lot of transaction processing in procedures and functions. Depending on the time available to the forensic examiner, this might take too long to execute. It is also in contrast to the goal of being a quicker test to establish trigger involvement.

8.5 Prototype Test Details

Both evaluated databases were populated with different test triggers, procedures and functions that all referenced the same database table. Care was taken to ensure that there was at least one test trigger of every category that the database supported.

Table 8.1: Test Database Objects. SQL Server

Object Name	**Object Type**	**Reference**
test_func	Function	Direct
test_proc	Procedure	Direct
test_dml	Trigger	Direct
test_ddl	Trigger	Direct
test_login	Trigger	Direct
test_dml_again	Trigger	Indirect
test_ddl_again	Trigger	Indirect

Table 8.1 lists all the test objects that were placed in the SQL Server database. All three trigger categories are represented, where the "test_login" object is the non-data trigger. There is also a test function and a test procedure. All test objects that are

marked "Direct" contain SQL code that references a table called *dbo.Benefit*. The other two triggers call the test function and test procedure respectively. Some of these test triggers were already introduced earlier during the trigger implementation discussion in chapter 5 (see figures 5.1 to 5.3).

Informal testing was conducted to establish if the prototype implementation was able to find all the test triggers that referred to the object *dbo.Benefit*, whether directly, or indirectly via the function or procedure.

```
Menu
1 - List DBs/Schemas
2 - Change DB/Schema
3 - List triggers
4 - Search triggers for object
5 - Display trigger content
9 - Quit

Enter menu option : 4
Enter object name : Benefit
Triggers referring to object Benefit :
Direct - 503672842 test_login LOGON 2014-05-23 2014-05-26
Direct - 583673127 test_dml DELETE 2017-11-20 2017-11-20
Direct - 631673298 test_ddl DROP_TABLE 2017-11-20 2017-11-20
Indirect - 759673754 test_dml_again DELETE 2018-10-13 2018-10-13
Indirect - 823673982 test_ddl_again DROP_DATABASE 2018-10-13 2018-10-13
```

Figure 8.6: Prototype Search Result. SQL Server.

Figure 8.6 shows the result for a test run performed on the SQL Server database. The prototype menu option "4" was selected and the object name "Benefit" was provided to the search algorithm. Not only were all the test triggers listed in table 8.1 correctly returned, but how they reference the specified database object was also correctly recognised.

The test setup for the Oracle database was very similar and produced the same results. Therefore, no separate discussion for the Oracle testing is provided.

8.6 Conclusion

The prototype demonstrated that the algorithm works in principle. In a simple test database the prototype was able to identify all the triggers associated with a specific

database object. However, various challenges where identified that the prototype might encounter in real-world databases. These would require mainly changes in the implementation, rather than to the algorithm itself. For the prototype to become a usable tool for the forensic examiner, a number of the identified challenges first need to be resolved, depending on which databases are targeted.

Thus far, this dissertation has concentrated on forensic examination of relational databases in general and particularly on how triggers can impact acquisition and analysis. However, the increasing adoption of databases built on the NoSQL model to supplement, and in some cases even replace, relational databases requires a closer look. The next chapter takes a brief excursion into the field of NoSQL databases and explores if forensic attribution can be performed analogous to relational databases.

Chapter 9

Forensic Examination of NoSQL Databases

Thus far, this dissertation has focused on forensic aspects of relational databases. However, a new generation of databases, known collectively as NoSQL databases, have started to make their appearance in current computer systems. These databases are either used to supplement the usage of relational databases or even completely replace them in certain environments. Thus, they have become forensically important and can no longer be ignored when studying and performing database forensics.

This chapter focuses on the presence of database triggers and the availability of certain security features in NoSQL databases. How they impact forensic examinations in general, and forensic attribution in particular, is explored. This chapter is based on work from a previously published paper (Hauger & Olivier, 2018).

Section 9.1 motivates the increasing forensic importance of NoSQL databases. In section 9.2, some detailed background information on some popular NoSQL databases is provided. Section 9.3 then surveys the same popular NoSQL databases regarding the availability of triggers, authentication, authorisation and logging. Thereafter, the results of this survey are analysed in section 9.4. Section 9.5 then discusses the implications of the results on digital forensics and specifically forensic attribution. Section 9.6 concludes this chapter.

9.1 Survey Context

To achieve greater flexibility and easier usage, NoSQL databases make use of much simpler models compared to relational databases. This includes simplified data storage models (see chapter 2). The departure from SQL as the query language has the added consequence that many of the other features in the SQL standard (such as triggers) may also not be present. This chapter explores the implications these simpler models used in NoSQL databases have on the process of forensic attribution.

Performing forensic attribution in digital systems is difficult. This is because the actions that need to be attributed occurred in the digital world, but the actors that are ultimately responsible are located in the physical world. Since it is difficult to tie digital world events to the physical world events, the next best option is to tie the actions in question to a program or process inside the digital realm (Cohen, 2009).

However, even tying actions to processes can be difficult without enough information sets that can be correlated to form a consistent chain of events (Cohen, 2009). Relational databases provide one such set of information in the form of traces in various log files and in system tables (Olivier, 2009). 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 part of the security model of all relational databases that follow the SQL standard. Consequently, this chapter 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 (Edlich, 2017) 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 some NoSQL DBMSs were chosen of which the trigger, 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 (see chapter 2).

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 (DB-Engines, 2017a). 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 examiner.

To study the trigger and security features, the official documentation of the latest version of the selected NoSQL DBMS as found published on the manufacturer's website was used (Apache Cassandra, 2016; MongoDB Inc., 2017b; Neo4j, 2017e; Redis Labs, 2017a). 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. Therefore, a discussion on distributed log files and the added complexities falls outside the scope of this survey.

9.2 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) (DB-Engines, 2017a). The top five thus represent all four NoSQL database types introduced in chapter 2. 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.

9.2.1 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. (Harris, 2013).

A record in MongoDB is a document, which is a data structure composed of field and value pairs. MongoDB documents are similar to JSON objects. The values of fields can 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. Finally, dynamic schemas support fluent polymorphism (MongoDB Inc., 2017a).

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, its 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 (MongoDB Inc., 2017a).

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 (MongoDB Inc., 2017a).

Prominent users of MongoDB include Metlife, Expedia, Ebay, SAP, SAGE, KPMG and Forbes (MongoDB Inc., 2017c).

9.2.2 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 (Hamilton, 2008). 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 (DataStax, 2017). 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 emphasises de-normalisation through CQL features such as 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 makes 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 (DataStax, 2017).

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 (DataStax, 2017). Failed nodes in a cluster can be replaced with no downtime.

Prominent users of Cassandra include CERN, Netflix, Reddit and eBay (Apache Cassandra, 2017).

9.2.3 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 (Sanfilippo, 2010). In 2015, Salvatore Sanfilippo joined Redis Labs which now sponsors development.

Redis maps keys to different types of values. It not only supports plain 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 such as 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 (Redis Labs, 2017b).

Redis is written in ANSI C and works in most POSIX systems such as Linux, various BSD operating systems and OS X without external dependencies. It works with an in-memory dataset, which can be optionally be persisted 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 (Redis Labs, 2017c).

9.2.4 Neo4j

Neo4j is an open-source graph database management system that is an ACID-compliant transactional database with native graph storage and processing (Neo4j, 2017a). 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[1]) through a transactional HTTP endpoint, or through the binary "bolt" protocol.

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 (Neo4j, 2017d).

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

[1]Not related to Cassandra Query Language (CQL)

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 (Neo4j, 2017a).

Prominent users of Neo4j include Walmart, Monsanto and Ebay (Neo4j, 2017c).

9.3 NoSQL 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: triggers, authentication, authorisation and logging. The official documentation was used as far as possible, but in some cases document gaps 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[2]. These additional features include enhancements and additions to authentication, authorisation and logging.

The results for the four 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.

9.3.1 MongoDB

Unless otherwise indicated, this section uses the official MongoDB documentation (MongoDB Inc., 2017b) to paraphrase the relevant information to indicate the availability of the surveyed features.

[2]See for example Neo4j Editions (Neo4j, 2017b)

MongoDB does not provide or support trigger functionality. A number of external add-on solutions have been developed by various MongoDB users to emulate specific trigger-like functionality (Gino, 2018; Torchlight Software, 2018). All these approaches make use of a transactional log file that is maintained when MongoDB replication is enabled.

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 Mechanism (SCRAM) family that uses the SHA-1 hash function. It is a mechanism for authenticating users with passwords and defined by the Internet Engineering Task Force (IETF) in the Request for Comments (RFC) 5802 standard (Newman, Menon-Sen, Melnikov, & Williams, 2010).

MongoDB employs role-based access control to govern access to a MongoDB system. A user is granted 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 needs to 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 and superuser. 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 and queries. 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.

9.3.2 Cassandra

Unless otherwise indicated, this section uses the official Cassandra documentation (Apache Cassandra, 2016) to paraphrase the relevant information to indicate the availability of the surveyed features.

Cassandra does provide trigger functionality that is similar to the SQL standard DML trigger. Triggers can be attached to tables and removed from them with the CREATE TRIGGER and DROP TRIGGER CQL commands. However, the trigger logic itself is not defined in the CQL but exists outside the database. The trigger invokes external Java code that needs to conform to a specific interface (Vajda, 2018).

The Java code is invoked before the DML statement is executed, thus making Cassandra triggers BEFORE statement-level triggers. The Java code is provided with all the intended data changes to the table in a mutation object. The Java code has the ability

to manipulate the given mutation object. Since any intended data changes will only be applied to the database after the trigger has executed, the trigger can be converted to an INSTEAD OF trigger.

The Java code also has the ability to add other database table changes to the mutation object. This gives the Cassandra trigger the ability to operate similar to a standard SQL trigger: a change of data in one table can be used to affect changes in other tables. Besides standard operations, the triggers can also be used for auditing, data transfer and communication purposes.

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 plain username/password authentication.

Cassandra uses a role-based access control framework, but provides no fixed or predefined roles. Cassandra roles do have a login property and a superuser property. The default Cassandra user has these properties set, so that 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 and 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 (Oracle, 2006), where a lower level of messages will include all the higher level ones as well. For example, the INFO level will include messages 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.

9.3.3 Redis

Unless otherwise indicated, this section uses the official Redis documentation (Redis Labs, 2017a) to paraphrase the relevant information to indicate the availability of the surveyed features.

Redis does not provide or support trigger functionality. However, Redis does provide a publish/subscribe messaging platform, which clients can connect to, in order to send and receive event notifications. When keyspace notifications are enabled in Redis, all events that effect key data will be published on this platform (Redis Labs, 2018). Since

only the key name and the event performed on the key are published, this allows only uses such as communication and limited auditing.

Since Redis is an in-memory database, it 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 needs to authenticate itself by sending the AUTH command, followed by the password. The AUTH command, such as 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 (Seguin, 2012).

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 rebuild the state after the database was shut down 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.

9.3.4 Neo4j

Unless otherwise indicated, this section uses the official Neo4j documentation (Neo4j, 2017e) to paraphrase the relevant information to indicate the availability of the surveyed features.

Neo4j does not provide SQL standard trigger functionality. It does, however, provide the ability to invoke actions based on data transactions via the TransactionEventHandler mechanism (Marzi, 2018). This allows the invocation of external Java code that can interrogate the transaction data and perform any action possible from the Java code. Uses for this ability include auditing, data transfer and communication.

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.

9.4 Discussion

In this section, the results from the feature survey in the previous section are discussed using two summary tables. The next section then discusses the implications of these results on forensic attribution.

Table 9.1: NoSQL Database Features

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

Table 9.1 summarises the results from the survey of triggers, access control and logging of the selected NoSQL databases. The first result presented by the summary, indicates that only Cassandra provides a native trigger mechanism that is comparable to a trigger as specified in the SQL standard. All the other surveyed NoSQL databases do not have native trigger mechanisms.

The second result this summary shows, is that all of the surveyed NoSQL databases do support authentication. However, the third 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 plain 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 these databases, all clients have the same full control over all database operations once they have been authenticated.

The fourth result that the summary in Table 9.1 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, such as 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 9.2: Features Enabled by Default

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

Table 9.2 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.

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 chosen 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 9.2 also shows that logging is not enabled by default on some databases. 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, it is not a great loss that logging is not enabled by default. 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.

9.5 Forensic Implications

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

9.5.1 Triggers

The survey found that only Cassandra provides native trigger functionality. The other three surveyed NoSQL databases do not have native triggers and can only emulate a particular subset of trigger capabilities via other mechanisms and add-on solutions. Cassandra only provides one of the three types of triggers identified in chapter 5: DML triggers.

The immediate implication of this finding is that forensic acquisition and analysis of these surveyed NoSQL databases is not impacted by triggers as was the case with certain relational databases. This is because these NoSQL databases do not provide non-data triggers.

Due to the presence of DML triggers in Cassandra, the process of forensic attribution could be impacted in the same way as for relational databases. The Java interface used by the trigger implementation allows it to behave as both a BEFORE trigger and an INSTEAD OF trigger. This means that actions of both commission and omission are possible for a Cassandra trigger.

When these trigger actions need to be attributed, they could be incorrectly attributed to the initiator of the original trigger action. Therefore, the forensic examiner needs to be aware of the presence of triggers. As with relational databases, the first step would be to determine if triggers are present in the Cassandra database under investigation. This could be achieved similarly to relational databases by executing a CQL query on the necessary tables.

If triggers are found to be present in the database, the next step would be to establish if any of those triggers could have influenced the data being examined. The idea of the trigger identification algorithm is still valid, however, the process will have to change quite a bit for the Cassandra trigger implementation. This is because the trigger implementation is located outside the database in a compiled Java class.

Although the other surveyed NoSQL databases do provide some trigger-like capabilities, these capabilities are limited to sending information out of the database. This information is used by other applications and systems for the purposes of auditing, data transfer and communication. Since any changes affected by the processing of this information occur outside of the primary database, they do not have an impact on the forensic examination of the primary database itself.

With some of the mechanisms, it might be theoretically possible to connect back to the primary database and affect data changes there. However, these changes would be made with a different connection and transaction context and one would expect this to be evident in some available trace. This is very different from a trigger, which would perform the additional or alternative data changes entirely inside the database within the same transaction.

9.5.2 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 plain password based access. From a security standpoint, this elementary authentication layer provides an improvement over having no authentication, but from a 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.

9.5.3 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.

Therefore, 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 examiner 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 (Adedayo & Olivier, 2015).

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 (Ellis, 2011). 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 examiner 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.

9.6 Conclusion

This chapter examined NoSQL databases in order to determine the presence of triggers and the availability of certain security features. The purpose of this examination was to ascertain the impact that these features have on performing forensic examinations in general, and specifically forensic attribution. A survey of four top ranked NoSQL databases was performed to determine what features they provide. The survey specifically looked at the areas of triggers, authentication, authorisation and logging.

The survey found that only the Cassandra NoSQL database provides trigger functionality. However, only DML triggers are provided. The forensic examiner therefore needs to take trigger actions into account when performing forensic attribution. Forensic acquisition and analysis of these NoSQL databases are, however, not impacted by triggers.

It was also found that 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.

Chapter 10

Conclusions

Forensic attribution is a lesser studied area of digital forensics. Particularly in the area of database forensics, this forensic process has not received much attention. Since forensic attribution is a difficult undertaking, the specific challenges presented by databases need to be known and understood.

This chapter presents a summary of the attribution challenges that this research identified and studied in further detail. Certain aspects that were out of scope for this research, and other areas that this research did not address, are also listed for possible future work.

Section 10.1 summarises the findings and conclusions from this research. The various contributions made by the research are highlighted in section 10.2. In section 10.3, some suggestions for future research are provided. These include areas that fell outside the scope of this research.

10.1 Summary of Conclusions

The main research question that this dissertation set out to answer was the following: What circumstances can prevent the correct attribution of actions performed in a database?

The research conducted in this dissertation identified and confirmed the following two challenges:

- The presence of database triggers.
- Insufficient access control and auditing traces.

The first identified challenge presented a completely new research aspect and consequently the scope of this challenge was extended to a more general research question: Do database triggers interfere with forensic examinations? This more general research question was divided into two parts that each target different processes in the forensic examination procedure.

The first focus point was the forensic acquisition process and the forensic analysis process. The specific research question for this focus point was formulated as follows: Do the current forensic acquisition and analysis processes make provision for the presence of triggers during a forensic examination of a database?

The research identified a group of triggers that it classified as non-data triggers. This class of trigger does not require any data changes in order to be activated. Therefore, triggers of this class have the potential to contaminate the data in a database by inconspicuous operations, such as connecting to the database, or by starting it up and shutting it down.

The research showed that the way databases are acquired and analysed during a forensic examination will make use of these inconspicuous operations. Thus, databases may be contaminated by the very person attempting to acquire and analyse the database without affecting the integrity and/or original condition. It does not matter whether the triggers present in the database were placed for operational or malicious purposes.

The second focus point was the forensic interpretation process, specifically the forensic attribution activity. The specific research question for this focus point was formulated as follows: Can attribution be accurately performed when dealing with actions that include those performed by triggers?

The research established that triggers can introduce side-effects into the normal flow of operations. These side-effects include performing additional actions or preventing the completion of the triggering operations. Certain types of triggers can also manipulate or completely replace the original operation. This means what the original operation intended to do, and what actually happened, are not necessarily the same.

The research also identified that a trigger performs its actions with the same user credentials as the original operation that caused the trigger to fire. Some databases might log additional information with an operation to indicate that it was performed by a trigger. However, one cannot assume that such an extended log will be available to the forensic examiner.

Both these effects of triggers can lead to incorrect conclusions when attributing operations performed in the database. Modified and additional operations might be incorrectly attributed to a user when, in fact, triggers were responsible.

The second identified challenge is caused by two connected issues. The first issue relates to user authentication and authorisation. Without proper access control, it is not possible to obtain a list of users, who have access to the database. It is also not possible to narrow down the user list according to who has permission to perform specific database operations.

The second issue relates to auditing/logging features that are either not available or are not enabled. Without sufficient logging, it is not possible to retrieve or construct an audit trail of who accessed the database, what operations were being performed and by whom.

The research found that all studied NoSQL databases had deficiencies when it came to providing traces for forensic attribution of database operations. This situation is a particular occurrence in NoSQL databases. Relational databases, in general, follow the SQL standard, which prescribes a security model that includes access control.

Two of the studied NoSQL databases did have security features similar to those found in relational databases. However, the security features were not enabled and configured appropriately in default configuration mode. When attempting forensic attribution, the forensic examiner is thus completely reliant on the configuration that the database administrator performed on the particular database.

Furthermore, the two other NoSQL databases did not provide security features that left relevant traces. In those databases, the forensic examiner can thus only use traces from outside the database to help perform forensic attribution of the operations that occurred inside the database. This situation will consequently influence the attribution accuracy negatively.

10.2 Contributions

As part of this research to address the problem statement, a number of other contributions were made to the fields of databases and database forensics.

One of the aims of this research was to perform a comprehensive evaluation of the impact that database triggers have on forensic examinations of databases, and particularly forensic attribution. This required an exhaustive list of trigger types and features.

Therefore, an investigation was conducted to determine all the types of triggers that are defined in the SQL standard, as well as those found in various popular relational databases. This investigation provided a new classification of available database triggers based on type and functionality.

Since there was not enough related research material available for an in-depth literature review, a literature survey of all major database forensic research was conducted instead. A previous literature survey performed in 2008 provided the starting point for this survey.

The results of this literature survey not only provide a picture of the current state of database forensic research, but they also give an indication of the research trend in this field. The amount of information available is increasing as the field is maturing. The research output, however, remains constant, which is in contrast to the similarly maturing field of cloud forensics, where research output is increasing.

Database triggers were found to impact various phases of the forensic examination process of databases. Therefore, database triggers have to be considered during a forensic examination. However, a database could potentially contain hundreds or more triggers that would need to be analysed and only some of them might be relevant to the examination.

In order to assist the forensic examiner with this task, an algorithm was developed to help identify only triggers that are relevant to the specific data being examined. The application of this algorithm allows the forensic examiner to focus only on the identified triggers, instead of all the triggers present in the database.

Insufficient or lacking security features are one of the by-products of the simplified models used by the NoSQL databases. In order to quantify the impact of unavailable security features on forensic attribution, a survey to determine the access control and

logging capabilities of a number of popular NoSQL databases was conducted.

The results of this survey not only provide a comparison of the current availability of access control and logging features in these popular NoSQL databases, but they also indicate whether these features are enabled by default. This information can provide the forensic examiner with a reasonable grasp of what traces he can expect to find, or not find, when he needs to perform a forensic examination of one of these NoSQL databases.

10.3 Future Work

Some aspects of triggers were not considered in this research in order to allow the fundamentals around the usage of triggers to be established first. Since a foundation has now been laid, these additional aspects can now be studied and added on top of the foundation.

10.3.1 Nested Triggers

This research only considered triggers on a single level. However, it is possible that the action performed by one trigger becomes a new event of either the same trigger or another trigger. This process of a trigger initiating another trigger is called nesting. There are two types of nested triggers. If each trigger activates another different trigger without repetition, then the triggers are cascading triggers. However, should all the nested triggers form a cycle, or a single trigger activate itself again, then the triggers are considered recursive triggers.

Different DBMSs have various configuration options that determine if trigger nesting is allowed and how many levels of nesting can be performed. These settings would form a suitable starting point for extending the current research. They would determine if further trigger identification and analysis may be required.

10.3.2 Deleted Triggers

Finding a database to be trigger free during a forensic examination does not always imply that triggers were never present. There is a possibility that triggers were present

at an earlier time and have since been deleted. The implication of this possibility is that some actions may have been performed by the deleted triggers and could now be incorrectly attributed to the initiator of the trigger event.

Foul play in a breached database may be a possible reason for this situation. Malicious agents may have created and used the triggers for nefarious purposes. When the triggers had served their purpose, they were then deleted together with other traces to hide the intrusion and related activities.

However, databases are not static and evolve over time as the computer systems that use them, need to adapt to change. Transactions that were initially performed by triggers, might have been moved to the application for greater flexibility. Therefore, when the new application version was deployed, the now obsolete triggers were deleted.

From a forensic perspective, it would have been valuable to still keep the triggers in the database in a disabled state. However, accidental reactivation of the triggers might have been considered too great of a risk. This would have led to duplicate transactions being performed and damaged the integrity of the data.

References

Adedayo, O. M., & Olivier, M. S. (2014). Schema Reconstruction in Database Forensics. In G. Peterson & S. Shenoi (Eds.), *Advances in Digital Forensics X* (p. 101-116). Heidelberg, Germany: Springer.

Adedayo, O. M., & Olivier, M. S. (2015, March). Ideal log setting for database forensics reconstruction. *Digital Investigation*, *12*, 27-40.

Adelstein, F. (2006). Live forensics: diagnosing your system without killing it first. *Communications of the ACM*, *49*(2), 63-66.

Ahmed, I., Zoranic, A., Javaid, S., III, G. R., & Roussev, V. (2013). Rule-Based Integrity Checking of Interrupt Descriptor Tables in Cloud Environments. In G. Peterson & S. Shenoi (Eds.), *Advances in Digital Forensics IX* (p. 305-328). Heidelberg, Germany: Springer.

Apache Cassandra. (2016). *Apache Cassandra Documentation v3.2.* Retrieved from https://cassandra.apache.org/doc/latest/ (Accessed: April 2017)

Apache Cassandra. (2017). *Apache Cassandra.* Retrieved from https://cassandra.apache.org/ (Accessed: April 2017)

Atkinson, M., Bancilhon, F., DeWitt, D., Dittrich, K., Maier, D., & Zdonik, S. (1989, December). The Object-Oriented Database System Manifesto. In *Proceedings of the First International Conference on Deductive and Object-Oriented Databases* (p. 223-240). Kyoto, Japan.

Atzeni, P., Ceri, S., Paraboschi, S., & Torlone, R. (1999). *Database Systems concepts, languages and architectures.* London, England GB: McGraw-Hill International (UK) Limited.

Beall, J. (2014, November). *Google Scholar is Filled with Junk Science.* Retrieved

from http://scholarlyoa.com/2014/11/04/google-scholar-is-filled-with-junk-science/ (Accessed: May 2015)

Bhoedjang, R., van Ballegooij, A., van Beek, H., van Schie, J., Dillema, F., van Baar, R., ... Streppel, M. (2012, November). Engineering an online computer forensic service. *Digital Investigation*, *9*, 96-108.

Boebert, W. E. (2010, June 10–11,). A Survey of Challenges in Attribution. In *Proceedings of a Workshop on Deterring CyberAttacks: Informing Strategies and Developing Options for U.S. Policy* (p. 41-52). Washington, DC.

Burch, J. W., Stanford, N., & Majerus, P. W. (1978, February). Inhibition of Platelet Prostaglandin Synthetase by Oral Aspirin. *The Journal of Clinical Investigation*, *61*(2), 314-319.

Butler, J. M., & Hill, C. R. (2012, January). Biology and Genetics of New Autosomal STR Loci Useful for Forensic DNA Analysis. *Forensic Science Review*, *24*(1), 15-26.

Cale, C. M., Earll, M. E., Latham, K. E., & Bush, G. L. (2016, January). Could Secondary DNA Transfer Falsely Place Someone at the Scene of a Crime? *Journal of Forensic Sciences*, *61*(1), 196-203.

Carrier, B. D. (2006). Risks of live digital forensic analysis. *Communications of the ACM*, *49*(2), 56-61.

Casey, E. (2009). *Handbook of Digital Forensics and Investigation.* Burlington, MA: Elsevier Academic Press.

Ceri, S., Cochrane, R. J., & Widom, J. (2000). Practical Applications of Triggers and Constraints: Success and Lingering Issues. In *Proceedings of the 26th International Conference on Very Large Data Bases* (p. 254-262). Morgan Kaufman Publishers Inc.

Chamberlin, D. D., Astrahan, M. M., Blasgen, M. W., Gray, J. N., King, W. F., Lindsay, B. G., ... Yost, R. A. (1981, October). A History and Evaluation of System R. *Communications of the ACM*, *24*(10), 632-646.

Chan, Y., Ivanov, N., & Mueller, O. (2013). *Oracle to DB2 Conversion Guide: Compatibility Made Easy* (3rd ed.). IBM ITSO.

Cherry, D. (2012). *Securing SQL Server* (2nd ed.). Waltham, MA: Elsevier.

Chivers, H., & Hargreaves, C. (2011, April). Forensic data recovery from the Windows Search Database. *Digital Investigation*, *7*, 114-126.

Chung, H., Parka, J., Lee, S., & Kang, C. (2012, November). Digital forensic investigation of cloud storage services. *Digital Investigation*, *9*, 81-95.

Clark, D. D., & Landau, S. (2010, June 10–11,). Untangling Attribution. In *Proceedings of a Workshop on Deterring CyberAttacks: Informing Strategies and Developing Options for U.S. Policy* (p. 25-40). Washington, DC.

Codd, E. F. (1970, June). A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, *13*(6), 377-387.

Codd, E. F. (1979, December). Extending the Database Relational Model to Capture More Meaning. *ACM Transactions on Database Systems*, *4*(4), 397-434.

Cohen, F. B. (2009). *Digital Forensic Evidence Examination* (4th ed.). Livermore, CA: Fred Cohen & Associates.

Cohen, F. B. (2010, January 5–8,). Attribution of messages to sources in digital forensics cases. In *Proceedings of the 43rd Hawaii International Conference on System Sciences* (p. 4459-4468). Honolulu, HI.

DataStax. (2017). *Apache Cassandra 3.0 for DSE 5.0.* Retrieved from http://docs.datastax.com/en/cassandra/3.0/ (Accessed: April 2017)

DB-Engines. (2014). *DB-Engines Ranking of Relational DBMS.* Retrieved from http://db-engines.com/en/ranking/relational+dbms (Accessed: May 2014)

DB-Engines. (2017a). *DB-Engines Ranking.* Retrieved from https://db-engines.com/en/ranking (Accessed: April 2017)

DB-Engines. (2017b). *DB-Engines Ranking - Trend Popularity.* Retrieved from https://db-engines.com/en/ranking_trend (Accessed: April 2017)

DB-Engines. (2018). *DBMS popularity broken down by database model.* Retrieved from https://db-engines.com/en/ranking_categories (Accessed: May 2018)

Delport, W., & Olivier, M. S. (2012). Isolating Instances in Cloud Forensics. In G. Peterson & S. Shenoi (Eds.), *Advances in Digital Forensics VIII* (p. 187-200). Heidelberg, Germany: Springer.

Dykstra, J., & Sherman, A. T. (2012, August 6–8,). Acquiring forensic evidence from infrastructure-as-a-service cloud computing: Exploring and evaluating tools, trust,

and techniques. In *Proceedings of the Twelfth Annual DFRWS Conference* (p. S90-S98). Washington, DC, USA.

Dykstra, J., & Sherman, A. T. (2013, August 4–7,). Design and implementation of FROST: Digital forensic tools for the OpenStack cloud computing platform. In *Proceedings of the Thirteenth Annual DFRWS Conference* (p. S87-S95). Monterey, CA, USA.

Edlich, S. (2017). *NOSQL Databases.* Retrieved from http://nosql-database.org/ (Accessed: April 2017)

Eifrem, E. (2009, October). *Emil Eifrem on Twitter.* Retrieved from https://twitter.com/emileifrem/statuses/5200345765 (Accessed: April 2017)

Ellis, J. (2009, November). *The NoSQL Ecosystem.* Retrieved from https://blog.rackspace.com/nosql-ecosystem (Accessed: April 2017)

Ellis, J. (2011, October). *Leveled Compaction in Apache Cassandra.* Retrieved from http://www.datastax.com/dev/blog/leveled-compaction-in-apache-cassandra (Accessed: April 2017)

Elmasri, R., & Navathe, S. B. (1994). *Fundamentals of Database Systems* (2nd ed.). Redwood City, CA: The Benjamin/Cummings Publishing Company Inc.

Epstein, B. (2013, April). History of Sybase. *IEEE Annals of the History of Computing*, *35*(2), 31-41.

Eriksson, J. (1997, September 8–9,). Real-Time and Active Databases: A Survey. In *Second International Workshop on Active Real Time and Temporal Database Systems.* Como, Italy.

Evans, E. (2009, October). *NoSQL: What's in a name?* Retrieved from http://blog.sym-link.com/2009/10/30/nosql_whats_in_a_name.html (Accessed: April 2017)

Fabbri, D., Ramamurthy, R., & Kaushik, R. (2013). SELECT triggers for data auditing. In *Proceedings of the 29th International Conference on Data Engineering* (p. 1141-1152). IEEE.

Fasan, O. M., & Olivier, M. S. (2012). Reconstruction in Database Forensics. In G. Peterson & S. Shenoi (Eds.), *Advances in Digital Forensics VIII* (p. 273-287). Heidelberg, Germany: Springer.

Federici, C. (2014, March). Cloud Data Imager: A unified answer to remote acquisition of cloud storage areas. *Digital Investigation*, *11*, 30-42.

Feinberg, D., Adrian, M., Heudecker, N., Ronthal, A. M., & Palanca, T. (2015, October). *Magic Quadrant for Operational Database Management Systems.* Retrieved from https://www.gartner.com/doc/3147919/magic-quadrant-operational-database-management (Accessed: April 2017)

Feuerstein, S., & Pribyl, B. (2014). *Oracle PL/SQL Programming* (6th ed.). Sebastopol, CA: O'Reilly Media Inc.

Fool Moon. (2014). *Windows Forensic Toolchest.* (WFT). Retrieved from http://www.foolmoon.net/security/wft/ (Accessed: May 2017)

Fowler, K. (2007a, April). *Forensic Analysis of a SQL Server 2005 Database Server.* Retrieved from http://www.sans.org/reading-room/whitepapers/application/forensic-analysis-sql-server-2005-database-server-1906 (Accessed: May 2015)

Fowler, K. (2007b). *A real world scenario of a SQL Server 2005 database forensics investigation.* Black Hat USA. Retrieved from https://www.blackhat.com/presentations/bh-usa-07/Fowler/Whitepaper/bh-usa-07-fowler-WP.pdf (Accessed: July 2014)

Fowler, K. (2009). *SQL Server Forenisc Analysis.* London, Great Britain: Pearson Education.

Fruhwirt, P., Kieseberg, P., Schrittwieser, S., Huber, M., & Weippl, E. (2012, August 20–24,). InnoDB Database Forensics: Reconstructing Data Manipulation Queries from Redo Logs. In *Proceedings of the Seventh International Conference on Availability, Reliability and Security* (p. 625-633). Prague, Czech Republic.

Garfinkel, S. L. (2010). Digital forensics research: The next 10 years. *Digital Investigation*, *7*(Supplement), S64-S73.

Gino, I. (2018). *MongoDB Triggers.* Retrieved from https://www.npmjs.com/package/mongo-triggers (Accessed: May 2018)

Google. (2018). *How Trends data is adjusted.* Retrieved from https://support.google.com/trends/answer/4365533?hl=en&ref_topic=6248052 (Accessed: September 2018)

Grimes, C. (2010, June). *Our new search index: Caffeine.* Retrieved from http://googleblog.blogspot.com/2010/06/our-new-search-index-caffeine.html (Accessed: May 2015)

Gulutzan, P., & Pelzer, T. (2003). *SQL Performance Tuning.* Boston, MA: Addison-Wesley Professional.

Haigh, T. (2011, April). Charles W. Bachman: Database Software Pioneer. *IEEE Annals of the History of Computing*, *33*(4), 70-80.

Hale, J. S. (2013, October). Amazon Cloud Drive forensic analysis. *Digital Investigation*, *10*, 259-265.

Hamilton, J. (2008, July). *Facebook Releases Cassandra as Open Source.* Retrieved from http://perspectives.mvdirona.com/2008/07/facebook-releases-cassandra-as-open-source/ (Accessed: April 2017)

Harbour, N. (2002). *dcfldd.* Department of Defense Computer Forensics Lab. Retrieved from http://dcfldd.sourceforge.net/ (Accessed: May 2017)

Hargreaves, C., & Chivers, H. (2008). Recovery of Encryption Keys from Memory Using a Linear Scan. In *Proceedings of the Third International Conference on Availability, Reliability and Security* (p. 1369-1376). IEEE.

Harris, D. (2013, August). *10gen embraces what it created, becomes MongoDB Inc.* Retrieved from https://gigaom.com/2013/08/27/10gen-embraces-what-it-created-becomes-mongodb-inc/ (Accessed: April 2017)

Hauger, W. K., & Olivier, M. S. (2015a). Determining trigger involvement during Forensic Attribution in Databases. In G. Peterson & S. Shenoi (Eds.), *Advances in Digital Forensics XI* (p. 163-177). Heidelberg, Germany: Springer.

Hauger, W. K., & Olivier, M. S. (2015b, June). The Impact of Triggers on Forensic Acquisition and Analysis of Databases. *SAIEE Africa Research Journal*, *106*(2), 64-73.

Hauger, W. K., & Olivier, M. S. (2015c, August 12–13,). The state of database forensic research. In *Proceedings of the 2015 Information Security for South Africa Conference* (p. 1-8). Johannesburg, South Africa.

Hauger, W. K., & Olivier, M. S. (2018, June). NoSQL Databases: Forensic Attribution Implications. *SAIEE Africa Research Journal*, *109*(2), 118-131.

Higgins, K. J. (2014, September). *Apple Not Hacked In Celebrity Nude Photo Breaches.* Retrieved from http://www.darkreading.com/cloud/apple-not-hacked-in-celebrity-nude-photo-breaches/d/d-id/1306906 (Accessed: May 2015)

IBM. (2012). *CREATE TRIGGER.* DB2 reference information. Retrieved from http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/index.jsp?topic=/com.ibm.db2z10.doc.sqlref/src/tpc/db2z_sql_createtrigger.htm (Accessed: May 2014)

ISO/IEC JTC1/SC27. (2015, March). *Information Technology – Security Techniques – Incident Investigation Principles and Processes* (ISO/IEC No. 27043:2015). Geneva, Switzerland: International Standards Organization and International Electrotechnical Commission.

ISO/IEC JTC1/SC32. (2008, July). *Information Technology – Database Languages – SQL – Part 4: Persistent Stored Modules* (ISO/IEC No. 9075-4:2008). Geneva, Switzerland: International Standards Organization and International Electrotechnical Commission.

ISO/IEC JTC1/SC32. (2011, December). *Information Technology - Database Languages – SQL – Part 2: Foundation* (ISO/IEC No. 9075-2:2011). Geneva, Switzerland: International Standards Organization and International Electrotechnical Commission.

Jones, A. (2005). *SQL Functions Programmer's Reference.* Indianapolis, IN: Wiley Publishing Inc.

jTDS Project, T. (2018). *jTDS JDBC Driver.* Retrieved from http://jtds.sourceforge.net/ (Accessed: April 2018)

Juola, P. (2008, March). Authorship Attribution. *Foundations and Trends in Information Retrieval*, *1*(3), 233-334.

Kelley, H. H. (1973, February). The processes of causal attribution. *American Psychologist*, *28*(2), 107-128.

Kessler, G. (2014, December). *File Signatures Table.* Retrieved from http://www.garykessler.net/library/file_sigs.html (Accessed: May 2015)

Khanuja, H. K., & Adane, D. (2012). A framework for database forensic analysis.

Computer Science and Engineering, *2*(3).

Kornbrust, A. (2005, April). *Database rootkits.* Black Hat Europe. Retrieved from http://www.red-database-security.com/wp/db_rootkits_us.pdf (Accessed: May 2014)

Krebs, B. (2015, January). *Lizard Stresser Runs on Hacked Home Routers.* Retrieved from http://krebsonsecurity.com/2015/01/lizard-stresser-runs-on-hacked-home-routers/ (Accessed: May 2015)

Lazenby, A., & Lindquist, M. T. (2007, May). Innovative From The Start, Oracle Looks To The Future. *Oracle Profit Magazine*, *12*(2), 26-33.

Lee, M., & Bieker, G. (2009). *Mastering SQL Server 2008.* Indianapolis, IN: Wiley Publishing Inc.

Leimich, P., Harrison, J., & J.Buchanan, W. (2016, September). A RAM triage methodology for Hadoop HDFS forensics. *Digital Investigation*, *18*, 96-109.

Litchfield, D. (2007). *The Oracle Hacker's Handbook: Hacking and Defending Oracle.* Indianapolis, IN: Wiley Publishing Inc.

Liu, C., Singhal, A., & Wijesekera, D. (2017). Identifying Evidence for Cloud Forensic Analysis. In G. Peterson & S. Shenoi (Eds.), *Advances in Digital Forensics XIII* (p. 111-130). Heidelberg, Germany: Springer.

Lubaale, E. C. (2015, June). Bokolo v S 2014 (1) SACR 66 (SCA). *SA Crime Quarterly*(52), 39-47.

Martini, B., & Choo, K. R. (2012, November). An integrated conceptual digital forensic framework for cloud computing. *Digital Investigation*, *9*, 71-80.

Martini, B., & Choo, K. R. (2013, December). Cloud storage forensics: ownCloud as a case study. *Digital Investigation*, *10*, 287-299.

Marzi, M. D. (2018). *Triggers in Neo4j.* Retrieved from https://dzone.com/articles/triggers-neo4j (Accessed: May 2018)

Mendelsohn, A. (2013, April). The Oracle Story: 1984-2001. *IEEE Annals of the History of Computing*, *35*(2), 10-23.

Microsoft. (2012a). *CREATE TRIGGER.* Data Definition Language (DDL) Statements. Retrieved from http://msdn.microsoft.com/en-us/library/ms189799.aspx (Accessed: May 2014)

Microsoft. (2012b). *Logon Triggers.* Database Engine Instances (SQL Server). Retrieved from http://technet.microsoft.com/en-us/library/bb326598.aspx (Accessed: May 2014)

Microsoft. (2015). *Diagnostic Connection for Database Administrators.* Database Engine Instances (SQL Server). Retrieved from https://msdn.microsoft.com/en-us/library/ms189595.aspx (Accessed: May 2017)

MongoDB Inc. (2017a). *Introduction to MongoDB.* Retrieved from https://docs.mongodb.com/manual/introduction/ (Accessed: April 2017)

MongoDB Inc. (2017b). *The MongoDB 3.4 Manual.* Retrieved from https://docs.mongodb.com/manual/ (Accessed: April 2017)

MongoDB Inc. (2017c). *Our Customers — MongoDB.* Retrieved from https://www.mongodb.com/who-uses-mongodb (Accessed: April 2017)

Morgenstern, M. (1983, October). Active Databases as a Paradigm for Enhanced Computing Environments. In *Proceedings of the Ninth International Conference on Very Large Data Bases* (p. 34-42). Florence, Italy.

Mullich, J. (2011, January). *16 Ways The Cloud Will Change Our Lives.* The Wall Street Journal. Retrieved from http://online.wsj.com/ad/article/cloudcomputing-changelives (Accessed: May 2015)

National Research Council. (1992). *DNA Technology in Forensic Science* (First ed.). Washington, DC: The National Academies Press.

National Research Council. (1996). *The Evaluation of Forensic DNA Evidence* (First ed.). Washington, DC: The National Academies Press.

Neo4j. (2017a). *Chapter 1. Introduction.* Retrieved from https://neo4j.com/docs/operations-manual/current/introduction/ (Accessed: April 2017)

Neo4j. (2017b). *Compare Neo4j Editions.* Retrieved from https://neo4j.com/editions/ (Accessed: April 2017)

Neo4j. (2017c). *Neo4j Customers.* Retrieved from https://neo4j.com/customers/ (Accessed: April 2017)

Neo4j. (2017d). *Neo4j Cypher Refcard 3.1.* Retrieved from https://neo4j.com/docs/cypher-refcard/current/ (Accessed: April 2017)

Neo4j. (2017e). *The Neo4j Operations Manual v3.1.* Retrieved from https://neo4j

.com/docs/operations-manual/current/ (Accessed: April 2017)

Newman, C., Menon-Sen, A., Melnikov, A., & Williams, N. (2010, July). *Salted Challenge Response Authentication Mechanism (SCRAM) SASL and GSS-API Mechanisms* (RFC No. 5802). RFC Editor. Internet Requests for Comments. Retrieved from http://www.rfc-editor.org/rfc/rfc5802.txt (Accessed: November 2017)

Oestreicher, K. (2014, August 3–6,). A forensically robust method for acquisition of iCloud data. In *Proceedings of the Fourteenth Annual DFRWS Conference* (p. S106-S113). Denver, CO, USA.

Olivier, M. S. (2009, March). On metadata context in Database Forensics. *Digital Investigation*, *5*, 115-123.

Olson, M. A., Bostic, K., & Seltzer, M. (1999, June 6–11,). Berkeley DB. In *Proceedings of the 1999 USENIX Annual Technical Conference.* Monterey, CA.

Oracle. (1998a). *Java SE Technologies - Database.* Retrieved from http://www.oracle.com/technetwork/java/javase/jdbc/index.html (Accessed: October 2015)

Oracle. (1998b). *JDBC Overview.* Retrieved from http://www.oracle.com/technetwork/java/overview-141217.html (Accessed: October 2015)

Oracle. (2006). *Java Logging Package.* Retrieved from https://docs.oracle.com/javase/6/docs/api/java/util/logging/package-summary.html (Accessed: April 2017)

Oracle. (2006). *Oracle Community.* Retrieved from https://community.oracle.com/community/developer/search.jspa?peopleEnabled=true&userID=&containerType=&container&q=select+trigger (Accessed: May 2014)

Oracle. (2009a). *CREATE TRIGGER Statement.* Database PL/SQL Language Reference 11g Release 2 (11.2). Retrieved from http://docs.oracle.com/cd/E11882_01/appdev.112/e17126/create_trigger.htm (Accessed: May 2014)

Oracle. (2009b). *PL/SQL Triggers.* Database PL/SQL Language Reference 11g Release 2 (11.2). Retrieved from http://docs.oracle.com/cd/E11882_01/appdev.112/e17126/triggers.htm (Accessed: May 2014)

Oracle. (2018, April). *Oracle JDBC FAQ.* Retrieved from http://www.oracle.com/technetwork/database/enterprise-edition/jdbc-faq-090281.html (Ac-

cessed: April 2018)

Oracle Corp. (2015). *CREATE TRIGGER Syntax.* MySQL 5.7 Reference Manual. Retrieved from http://dev.mysql.com/doc/refman/5.7/en/create-trigger.html (Accessed: May 2014)

Osborne, C. (2014, February 13,). *How hackers stole millions of credit card records from Target.* ZDNet. Retrieved from http://www.zdnet.com/how-hackers-stole-millions-of-credit-card-records-from-target-7000026299/ (Accessed: May 2014)

O'Shaughnessy, S., & Keane, A. (2013). Impact of Cloud Computing on Digital Forensic investigations. In G. Peterson & S. Shenoi (Eds.), *Advances in Digital Forensics IX* (p. 291-303). Heidelberg, Germany: Springer.

Pereira, M. T. (2009, March). Forensic analysis of the Firefox 3 Internet history and recovery of deleted SQLite records. *Digital Investigation*, *5*, 93-103.

Pichan, A., Lazarescu, M., & Soh, S. T. (2015, June). Cloud forensics: Technical challenges, solutions and comparative analysis. *Digital Investigation*, *13*, 38-57.

Pieterse, H., & Olivier, M. S. (2012). Data Hiding Techniques for Database Environments. In G. Peterson & S. Shenoi (Eds.), *Advances in Digital Forensics VIII* (p. 289-301). Heidelberg, Germany: Springer.

Pollitt, M. M. (2007). An Ad Hoc Review of Digital Forensic Models. In *Proceedings of the Second International Workshop on Systematic Approaches to Digital Forensic Engineering* (p. 43-54). Morgan Kaufman Publishers Inc.

Pollitt, M. M. (2013). *The Hermeneutics Of The Hard Drive: Using Narratology, Natural Language Processing, And Knowledge Management To Improve The Effectiveness Of The Digital Forensic Process* (Doctoral thesis). College of Arts and Humanities, University of Central Florida, Department of English, Orlando, FL.

PostgreSQL Group. (2013). *CREATE TRIGGER.* PostgreSQL 9.3.4 Documentation. Retrieved from http://www.postgresql.org/docs/9.3/static/sql-createtrigger.html (Accessed: May 2014)

Quick, D., & Choo, K. R. (2013a, June). Dropbox analysis: Data remnants on user machines. *Digital Investigation*, *10*, 3-18.

Quick, D., & Choo, K. R. (2013b, October). Forensic collection of cloud storage data:

Does the act of collection result in changes to the data or its metadata? *Digital Investigation*, *10*, 266-277.

Ramakrishnan, R., & Gehrke, J. (2003). *Database Management Systems.* New York City, NY: McGraw-Hill Education.

Ras, D., & Olivier, M. S. (2012). Finding File Fragments in the Cloud. In G. Peterson & S. Shenoi (Eds.), *Advances in Digital Forensics VIII* (p. 169-185). Heidelberg, Germany: Springer.

Redis Labs. (2017a). *Documentation.* Retrieved from https://redis.io/documentation (Accessed: April 2017)

Redis Labs. (2017b). *Introduction to Redis.* Retrieved from https://redis.io/topics/introduction (Accessed: April 2017)

Redis Labs. (2017c). *Who's using Redis?* Retrieved from https://redis.io/topics/whos-using-redis (Accessed: April 2017)

Redis Labs. (2018). *Redis Keyspace Notifications.* Retrieved from https://redis.io/topics/notifications (Accessed: May 2018)

Rid, T., & Buchanan, B. (2015, January). Attributing Cyber Attacks. *Journal of Strategic Studies*, *38*(1–2), 4-37.

Rid, T., & McBurney, P. (2012, February). Cyber-Weapons. *The RUSI Journal*, *157*(1), 6-13.

Robinson, I., Webber, J., & Eifrem, E. (2015). *Graph Databases* (Second ed.). Sebastopol, CA: O'Reilly Media, Inc.

Roussev, V., Ahmed, I., Barreto, A., McCulley, S., & Shanmughan, V. (2016, September). Cloud forensics – Tool development studies and future outlook. *Digital Investigation*, *18*, 79-95.

Roussev, V., Barreto, A., & Ahmed, I. (2016). API-Based Forensic Acquisition of Cloud Drives. In G. Peterson & S. Shenoi (Eds.), *Advances in Digital Forensics XII* (p. 213-235). Heidelberg, Germany: Springer.

Roussev, V., & McCulley, S. (2016, March 29–31,). Forensic analysis of cloud-native artifacts. In *Proceedings of the Third Annual DFRWS Europe Conference* (p. S104-S113). Lausanne, Switzerland.

Ruan, K., Carthy, J., Kechadi, T., & Baggili, I. (2013, June). Cloud forensics definitions

and critical criteria for cloud forensic capability: An overview of survey results. *Digital Investigation*, *10*, 34-43.

Ruan, K., Carthy, J., Kechadi, T., & Crosbie, M. (2011). Cloud Forensics. In G. Peterson & S. Shenoi (Eds.), *Advances in Digital Forensics VII* (p. 35-46). Heidelberg, Germany: Springer.

Ruan, K., James, J., Carthy, J., & Kechadi, T. (2012). Key Terms for Service Level Agreements to Support Cloud Forensics. In G. Peterson & S. Shenoi (Eds.), *Advances in Digital Forensics VIII* (p. 201-212). Heidelberg, Germany: Springer.

Sanfilippo, S. (2010, March). *VMware: the new Redis home.* Retrieved from http://oldblog.antirez.com/post/vmware-the-new-redis-home.html (Accessed: April 2017)

Seguin, K. (2012). *The Little Redis Book.* Self-Published.

Shamsi, J. A., Zeadally, S., Sheikh, F., & Flowers, A. (2016, October). Attribution in cyberspace: techniques and legal implications. *Security and Communication Networks*, *9*(15), 2886-2900.

Silberschatz, A., Korth, H. F., & Sudarshan, S. (2011). *Database System Concepts* (6th ed.). New York, NY: McGraw-Hill.

Simon, E., & Kotz-Dittrich, A. (1995). Promises and Realities of Active Database Systems. In *Proceedings of the 21th International Conference on Very Large Data Bases* (p. 642-653). Morgan Kaufman Publishers Inc.

Singhal, A. (2013, May). *A multi-screen and conversational search experience.* Retrieved from http://insidesearch.blogspot.com/2013/05/a-multi-screen-and-conversational.html (Accessed: May 2015)

Sippl, R. (2013, April). Informix: Information Management on Unix. *IEEE Annals of the History of Computing*, *35*(2), 42-53.

SQLite Cons. (2014a). *Appropriate Uses For SQLite.* Categorical Index Of SQLite Documents. Retrieved from http://www.sqlite.org/famous.html (Accessed: September 2014)

SQLite Cons. (2014b). *CREATE TRIGGER.* SQL As Understood By SQLite. Retrieved from http://www.sqlite.org/lang_createtrigger.html (Accessed: September 2014)

SQLite Cons. (2014c). *Well-Known Users of SQLite.* Categorical Index Of SQLite Documents. Retrieved from http://www.sqlite.org/famous.html (Accessed: September 2014)

Stonebraker, M., Held, G., Wong, E., & Kreps, P. (1976, September). The design and implementation of INGRES. *ACM Transactions on Database Systems*, *1*(3), 189-222.

Stonebraker, M., Rowe, L. A., Lindsay, B. G., Gray, J., Carey, M. J., Brodie, M. L., ... Beech, D. (1990, September). Third-Generation Database System Manifesto. *ACM SIGMOD Record*, *19*(3), 31-44.

Strauch, C. (2011). *NoSQL Databases.* Retrieved from http://www.christof-strauch.de/nosqldbs.pdf (Accessed: April 2017)

Sullivan, D. (2014). *NoSQL for Mere Mortals* (First ed.). Hoboken, NJ: Addison-Wesley Professional.

Sybase Inc. (2011). *Create Trigger.* Adaptive Server Enterprise 15.7: Reference Manual - Commands. Retrieved from http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.infocenter.dc36272.1570/html/commands/X19955.htm (Accessed: September 2014)

Thompson, W. C., Mueller, L. D., & Krane, D. E. (2012, December). Forensic DNA Statistics: Still Controversial in Some Cases. *Champion: The Official News Report of the National Association of Criminal Defense Lawyers*, 12-23.

Torchlight Software. (2018). *Mongo Watch.* Retrieved from https://www.npmjs.com/package/mongo-watch (Accessed: May 2018)

Trenwith, P., & Venter, H. (2015). Locating and Tracking Digital Objects in the Cloud. In G. Peterson & S. Shenoi (Eds.), *Advances in Digital Forensics XI* (p. 287-301). Heidelberg, Germany: Springer.

Turley, P., & Wood, D. (2009). *Beginning T-SQL with Microsoft SQL Server 2005 and 2008.* Indianapolis, IN: Wiley Publishing Inc.

Turner, C. R., Fuggetta, A., Lavazza, L., & Wolf, A. L. (1999). A conceptual basis for feature engineering. *The Journal of Systems and Software*, *49*(1), 3-15.

Vajda, V. (2018). *Cassandra to Kafka Data Pipeline Part 1.* Retrieved from https://www.smartcat.io/blog/2017/cassandra-to-kafka-data-pipeline-part-1/ (Ac-

cessed: May 2018)

Verschoor, R. (2007, January). *Login triggers in ASE 12.5+.* Retrieved from http://www.sypron.nl/logtrig.html (Accessed: September 2014)

Wallenius, M., Mayer, K., & Ray, I. (2006, January). Nuclear forensic investigations: Two case studies. *Forensic Science International*, *156*(1), 55-62.

Wheeler, D. A., & Larsen, G. N. (2003, October). *Techniques for Cyber Attack Attribution* (No. P-3792). Institute for Defense Analyses. VA.

Widom, J. (1994). Deductive and Active Databases: Two Paradigms or Ends of a Spectrum? In N. W. Paton & M. H. Williams (Eds.), *Rules in Database Systems* (p. 306-315). London, Great Britain: Springer.

Zawoad, S., & Hasan, R. (2015). A Trustworthy Cloud Forensics Environment. In G. Peterson & S. Shenoi (Eds.), *Advances in Digital Forensics XI* (p. 271-285). Heidelberg, Germany: Springer.

Appendix A

Acronyms

The following is a list of the important acronyms that were used in this dissertation.

ACM Association for Computing Machinery

ANSI American National Standards Institute

API Application Programming Interface

DBMS Database Management System

DDL Data Definition Language

DDoS Distributed Denial of Service

DFRWS Digital Forensics Research Workshop

DML Data Manipulation Language

DoS Denial of Service

EMI Electromagnetic Interference

GPS Global Positioning System

IEEE Institute of Electrical and Electronics Engineers

IETF Internet Engineering Task Force

IFIP	International Federation for Information Processing
IP	Internet Protocol
ISO	International Organization for Standardization
JDBC	Java Database Connectivity
JSON	JavaScript Object Notation
MAC	Media Access Control
MD5	Message Digest 5
ODBC	Open Database Connectivity
RAID	Redundant Array of Independent Disks
RFC	Request for Comments
SAN	Storage Area Network
SCRAM	Salted Challenge Response Authentication Mechanism
SHA1	Secure Hash Algorithm 1
SQL	Structured Query Language
XML	Extensible Markup Language

Appendix B

Derived Publications and Conference Papers

The following is a list of all the publications and conference papers that were derived from this dissertation.

- W.K. Hauger and M.S. Olivier (2014, August): "The role of triggers in database forensics", In *Proceedings of the 2014 Information Security for South Africa Conference*, pp. 1–7. Johannesburg, South Africa.
- W. Hauger and M. Olivier (2015): "Determining trigger involvement during forensic attribution in databases", In G. Peterson & S. Shenoi (Eds.), *Advances in Digital Forensics XI*, pp. 163–177. Heidelberg, Germany: Springer.
- W.K. Hauger and M.S. Olivier (2015, June): "The impact of triggers on forensic acquisition and analysis of databases", In *SAIEE Africa Research Journal, 106*(2), pp. 64–73.
- W.K. Hauger and M.S. Olivier (2015, August): "The state of database forensic research", In *Proceedings of the 2015 Information Security for South Africa Conference*, pp. 1–8. Johannesburg, South Africa.
- W.K. Hauger and M.S. Olivier (2017, August): "Forensic attribution in NoSQL

databases", In *Proceedings of the 2017 Information Security for South Africa Conference*. Johannesburg, South Africa.

- W.K. Hauger and M.S. Olivier (2018, June): "NoSQL Databases: Forensic Attribution Implications", In *SAIEE Africa Research Journal, 109*(2), pp. 118–131.