Digital Investigation 12 (2015) 27–40

Contents lists available at ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin

Ideal log setting for database forensics reconstruction

Oluwasola Mary Adedayo[*], Martin S. Olivier

ICSA Research Group, Computer Science, University of Pretoria, South Africa

A R T I C L E I N F O

Article history:
Received 6 June 2014
Received in revised form 27 November 2014
Accepted 7 December 2014
Available online 13 January 2015

Keywords:
Database management system
Database forensics
Digital forensics
Reconstruction
Ideal log setting

A B S T R A C T

The ability to reconstruct the data stored in a database at an earlier time is an important aspect of database forensics. Past research shows that the log file in a database can be useful for reconstruction. However, in many database systems there are various options that control which information is included in the logs. This paper introduces the notion of the ideal log setting necessary for an effective reconstruction process in database forensics. The paper provides a survey of the default logging preferences in some of the popular database management systems and identifies the information that a database log should contain in order to be useful for reconstruction. The challenges that may be encountered in storing the information as well as ways of overcoming the challenges are discussed. Possible logging preferences that may be considered as the ideal log setting for the popular database systems are also proposed. In addition, the paper relates the identified requirements to the three dimensions of reconstruction in database forensics and points out the additional requirements and/or techniques that may be required in the different dimensions.

Introduction

The need to perform the forensic investigation of a database is becoming increasingly important due to the fact that information relating to various investigations can often be found in a database. Even though a database may not be the focus of a crime, it often contains information that may assist in solving a crime. Database forensics is a branch of digital forensics that deals with the identification, preservation, analysis, and presentation of evidence from databases (Fowler, 2008). Although the digital forensics field has experienced a tremendous growth in the last decade (Garfinkel, August 2010), this cannot be said of database forensics. Despite the large amount of research in other related fields, very little has been done on database forensics (Olivier, 2009) and there is need for the development of techniques for an effective forensic investigation of databases.

An important aspect of database forensics deals with the ability to reconstruct the information stored in a database at an earlier time (Olivier, 2009; Fasan and Olivier, 2012c). Many of the approaches that have been considered in this regard and in the forensic investigation of various database systems (Fowler, 2008; Litchfield, March 2007a; Fasan & Olivier, 2012a, 2012c; Adedayo and Olivier, 2013) explore the use of the log files in different databases as a source of information for performing forensics analysis on the database. Most of the previous research also assume that the information needed is contained in the log files even though this may not always be the case.

Although most databases have the capability to log operations performed on them, the content of any log file is determined either by the logging preferences enabled by the user or by the default settings on the database

* Corresponding author.
E-mail addresses: madedayo@cs.up.ac.za (O.M. Adedayo), molivier@cs.up.ac.za (M.S. Olivier).

http://dx.doi.org/10.1016/j.diin.2014.12.002
1742-2876/© 2014 Elsevier Ltd. All rights reserved.

management system (DBMS). Unfortunately, the default setting on a database may not be adequate for reconstruction, and for database forensics in general since a setting that requires the least amount of space possible is often made the default setting.

In this paper, we introduce the notion of an *ideal log setting* – a combination of logging preferences on a database that allows an effective forensics analysis of the database, and specifically enables the reconstruction of the data previously stored on the database prior to various kinds of modifications or deletions. The paper provides a survey of the default logging preferences on some of the popular DBMSs. We also determine the information that should be included in an ideal log. This definition is then applied in specifying an ideal log setting for database forensics reconstruction in the selected databases. In order to confirm that the set of required information is indeed adequate, we focus on the three dimensions of database forensics and reflect on the usefulness of each information in the different dimensions. We note that although the ideas presented in this paper may be useful in terms of the forensics readiness of a database system, the focus is on ensuring that the information required for reconstruction during a database forensics investigation is available.

In Background Section, we briefly discuss the concepts and architecture of a DBMS as well as the concepts and dimensions of reconstruction in database forensics. A survey of the default log settings in six of the most popular DBMSs is given in Default Log Section. Ideal Log Settings Section give a detailed explanation of the information that needs to be logged in a database for effective reconstruction to be possible and describes how this log setting may be achieved in the six database systems. A reflection on the identified set of required information, in relation to the dimensions that exist in database forensics reconstruction is discussed in Database Forensics Reconstruction Dimensions and Log Files Section. In Conclusion and Future Work Section, we give the conclusion and highlight some of the future work relating to this paper.

Background

In this section, we review some of the concepts related to this work. We describe the purpose of logging in database management systems as well as the relevance of database log files in the reconstruction process of database forensics.

Logging in databases

Databases play a critical role in almost all areas where computers are used. It enables the storage of information and the use of queries in retrieving or updating of specific records. Databases provide various techniques for maintaining the integrity of stored data, ensuring the security of the database, and recovering from failures that may occur in a database. Many of these techniques rely on the use of logs maintained by the DBMS. Logs are used to keep a record of operations that affect the value of database items (Elmasri, 2008) and are flushed to log files on disk, usually before any associated data changes are made to data files. In the event of a failure, the recovery process of a database reads the log files and uses the log records in retrieving a consistent version of the database (Fang et al., 2011).

The overhead incurred in the process of writing log records may be seen as one of the major reasons why a DBMS allows various levels of logging in a database, in order to enable an administrator to decide how much information should be logged. However, a default set of logging preferences is always enabled if the desired preferences are not specified. The information logged when using the default preferences are different for various DBMSs. In Default Log Section we give a survey of the default setting in six database systems that have been selected in this work based on their popularity and divergence in usage. These database systems are MySQL, Microsoft SQL Server, Postgres, Oracle, DB2, and Sybase.

Concept of reconstruction in digital forensics

The process of reconstruction in a digital forensics investigation refers to the examination of digital data with the aim of identifying why it has its characteristics and the events that might have caused it occurrence. It also questions the source and the time of creation of a digital object. Reconstruction can be divided into the two major categories described below.

- Experimental reconstruction: this is focused on the testing of hypotheses about an investigation in order to confirm, refute or reveal more information about the hypotheses. Examples and applications of experimental reconstruction techniques can be found in Bevel and Gardner (2002), Casey (2011), Turvey (2002).
- Retrospective reconstruction: this involves an attempt to go backwards in time in order to determine the previous inputs and states that could have generated the available data. Examples of this reconstruction approach are discussed in Gladyshev & Patel (2004), Carrier (2006).

Regardless of the reconstruction approach used, digital data usually contains the following three types of information that can be explored for reconstruction (Casey, 2011; Pollitt, 2010):

1. Relational information: this includes information about the location of a piece of data and its relationship with other data.
2. Functional information: this describes the operation, purpose or capability of digital data in relation to the investigation being conducted.
3. Temporal information: this provides clues about the interaction of time and environment on the piece of data. This may include date-time stamps in log files or records about the last access or modification of a file.

Although the concept of reconstruction in the general field of digital forensics is often applied to identify events that might be responsible for an incident, its application in database forensics also involves the reconstruction of data that might have existed on a database at earlier time prior to some modifications or deletion (Olivier, 2009; Fasan and Olivier, 2012c). In the following section, we describe the use of the temporal, functional and relational information that can be extracted from database log files (about the data in a database) in the reconstruction of data previously stored or queries previously executed on the database.

Log files and database forensics reconstruction

Although only a little amount of work has been done on database forensics, much of the work that has been done explore the information available on a database log file due to the volume of relational, functional and temporal information that is often available in the log files about the data stored and queries executed on a database.

In a series of papers by Litchfield (Litchfield, March 2007a, 2007b, 2007c, April 2007, August 2007a, 2007b, November 2008), he described the technical details of the forensics examination of a Oracle databases and how the redo logs in Oracle can be used in locating dropped objects and recovering data in an Oracle database. Wright (Wright and Burleson, 2010) published a book that explains Oracle forensics and investigates the possibility of using Oracle LogMiner (Wright, 2005) as a forensic tool. In his book on SQL Server forensics, Fowler (2008) identified that the ability to reconstruct SQL statements previously executed during an incident is of great importance during database forensics. He described how the transaction log in SQL server database can be analyzed in order to reconstruct the data manipulation operations that were previously performed on the database.

In recent papers by Frühwirt et al. (2010, 2012), the content of the redo logs in InnoDB is also explored for the forensics analysis of the database. The authors describe how the data manipulation queries executed on an InnoDB database can be reconstructed from the log files during an investigation. Other recent work by Fasan and Olivier (Fasan & Olivier, 2012a, 2012c; Adedayo and Olivier, 2013) describe a database reconstruction algorithm and proves it to be correct. The reconstruction algorithm explores the query log in a database and reconstructs earlier instances of records or tables using the concept of inverse relational algebra and value blocks.

All the work described above explore the use of the log files on different databases for reconstruction and other aspects of database forensics. Many of them also assume that the logging of the required information is enabled on the database being investigated and that the log file can be analyzed in order to find relevant information. However, we posit that this may not always be the case since most DBMSs allow different levels of logging and usually include a default logging preference which may not be adequate for reconstruction should the need to investigate the database arise. This is due to the fact that the logging preferences selected on a database affect the volume and relevance of the information that may be available in its log during a forensic investigation. For example, the technique to be used in handling log rotation or what should be done when a log becomes full depends on the logging preferences and this affects how much of past information is available during an investigation.

The use of log files for database forensics reconstruction seems to be a unifying factor for the various dimensions of database forensics identified by Fasan and Olivier (2012b). The authors (Fasan and Olivier, 2012b) discuss the three dimensions that exist in database forensics research and show that even though the dimensions seem to be divergent, the diverging strands are in fact, related in that they form different dimensions of the same problem space. The three dimensions identified in Fasan and Olivier (2012b) are described as follows.

- Compromised database: this is a database where some of the metadata or some software of the DBMS have been modified by an attacker even though the database is still operational. A major concern of database forensics in this situation is that an investigator cannot trust the information provided by the database being investigated, including the information in the log files. This problem was also identified by Litchfield (April 2007) and Beyers et al. (2011). One major decision that has to be made when investigating a compromised database is whether to use the metadata as it occurs on the database being investigated or to try to get a clean copy of the DBMS (Olivier, 2009; Beyers et al., 2011).
- Damaged or destroyed database: this refers to databases where the data contained, data files or log files may have been modified, deleted or copied from their original locations into other places. These databases may or may no longer be operational depending on the extent of the damage done. Most of the research (Wright and Burleson, 2010; Wright, 2005; Pollitt, 2010; Stahlberg et al., 2007) in database forensics fall into this dimension. In order to carry out the forensics analysis of a damaged database, it may be necessary to employ techniques in file carving (Carrier, 2005) in regenerating destroyed files or underlying files in a database which may helpful in retrieving the data that are of interest.
- Modified database: this is a database which has not been compromised or damaged but has undergone changes due to normal business processes since an event of interest occurred. This dimension of database forensics is often of interest when a database is not directly involved in the incident being investigated but is used to store information that may assist in solving an incident. Fasan and Olivier (2012a), Fasan and Olivier (2012c) examine how the information in a modified database at an earlier time can be reconstructed even though several modifications of the data might have occurred. The authors explore the content of log files and introduced the notion of relational algebra log, value blocks and inverse relational algebra in developing a reconstruction algorithm for database forensics.

Although these dimensions of database forensics reconstruction are unified by the use of log files, the challenges faced in handling the log files in each category differs. In addition, the logging preferences that should be selected in databases that are susceptible to compromise or damage may be different from the preferences required for modified databases.

The purpose of this paper is to explore the default logging preferences on some of the popular databases (MySQL, Microsoft SQL Server, Postgres, Oracle, DB2, Sybase) and how the preferences affect the information needed for reconstruction during a database forensic investigation. The paper also determines the information that are important for reconstruction and the appropriate logging preferences that will enhance reconstruction in the selected databases. In addition, we consider the identified information requirements and logging preferences in relation to each dimension of database reconstruction earlier described. Specific requirements of the individual dimensions, necessary to ensure an effective forensics analysis and reconstruction are also discussed in the paper.

Default log

In this section, we consider six of the popular databases and describe their default logging preferences. Although the default preferences are often the same in different versions of many of the database systems considered, the specific versions of the systems studied in this paper are MySQL 5.6, SQL Server 2008 and 2012, Postgres 9.2, Oracle 11g release 1 (11.1), DB2 10.1, and Sybase IQ 15.1. In addition, the paper describes only the logs considered most useful for forensic analysis in many of the systems. The effect of the default logging preference on the information that may be available for reconstruction during an analysis of the database thereafter are also discussed in this section.

MySQL

MySQL is an open source relational database management system. The database software works as a client/server or embedded system with a multi-thread SQL server that supports various backends, client programs, administrative and other tasks. MySQL server is the main program in the DBMS that manages access to the data directory containing the databases, tables, and other information such as the log files (Oracle, 2013). MySQL server has five different log files, namely the error log, general query log, binary log, slow query log and the relay log. These logs store information about the activities taking place in the database.

The error log is used to store information about problems encountered when starting, running or stopping a MySQL server. Error logging is enabled by default on a MySQL database. The general query log is used to store information about the client connections that are established and SQL statements or queries received from the client. Queries written to this log are in the order in which they are received and this may be different from the order of their execution. All queries including statements that are only used to select or show data are stored when the general query log is enabled. The general query log is disabled by default. The slow query log is used to store queries that take more than a specified amount of time to execute (the default time is set as 10 μs). Similar to the general query log, statements are written to the slow query log file after their execution and before the release of any locks on the database. Thus, the order of the log entries may be different from the order in which the queries are executed. The slow query log is disabled by default.

The Binary log is used to store statements that change the data on a MySQL database. For example, it stores table creation operations and operations that change table data. The binary log also contains information about the duration of the queries that update data. Unlike the general query log, it does not store queries that do not modify data. Also, statements are logged after they are completed but before the database locks are released. Thus, the order of the statements in the logs is the same as the order of execution. The format of the statements stored in the binary log is dependent on the logging format enabled (row-based, statement-based or mixed base logging) on the database (Oracle, 2013). The binary log is disabled by default. Although enabling the binary log on MySQL database makes performance slightly slower, it is important for two major purposes; for the replication of data changes on slaves and for recovery of the current instance of a database from previous backup point (Oracle, 2013). The Relay log is used to store data changes received from a replication server master (Oracle, 2013). It also stores statements that describe database changes and has the same format as the binary log file. The relay logs are automatically deleted once all the events in the file have been executed and it is no longer needed. The relay log is not enabled in the default setting of the server.

Quite a large volume of information can be collected from MySQL server logs for an investigation, if the logs are enabled. Unfortunately, the default logging preference on MySQL disables all the logs except for the error log (in the Windows operating system) and even when the binary log is enabled, it is by default synchronized to disk at each write. In the event that the forensics analysis of a MySQL server is required, the error log contains virtually no information that can be useful for reconstruction or forensic investigation of the database in general. It is important that a logging preference that is suitable for database forensics be enabled by the user or administrator. In Ideal Log Settings Section, we describe a possible logging preference that allows an effective forensics analysis and the reconstruction of data on a MySQL database.

Microsoft SQL Server

The Microsoft SQL Server (henceforth referred to as SQL Server) is another relational database management system. An SQL Server installation contains a set of system databases as well as the user databases that are created as needed (Rankins et al., 2010). The user databases are created for data storage and manipulation by the user. Similar to the system databases, a user database consists of two types of operating system files: the data files and the log files. The data files are used to store database objects such as tables and procedures while the log files keep record of the events that occur on the database. There are four main types of logs[1] that are available on the SQL Server

[1] Other log files in Microsoft SQL Server include the SQL Server setup log and the SQL Server profiler log (Otey, 2006).

database: the Windows event log, SQL Server agent log, SQL Server error log and the transaction log.

The Windows event log contains three useful logs that can be used for troubleshooting the SQL Server errors; the application log which stores events that take place in the SQL Server and SQL Server agent), the security log that stores authentication information and the system log which stores service startup and shutdown information. The SQL Server agent log keeps record of any information, warning and error messages concerning the SQL Server agent and its operations. A new SQL Server agent log is created with a time stamp each time the server is started (Otey, 2006). The SQL Server error log works in the same way as the SQL Server agent log and is used to store information about errors that occur during the database operations. For example, if a procedure refuses to startup.

The transaction log is arguably the most important log file and a critical component of an SQL Server database because it keeps a record of all data modification queries executed on the database in the order in which they occur. The database ensures that data is written to the transaction log files before they are effected on the database, thus transaction logs are useful for recovery from failed transactions and retrieval of a consistent version of the database in the event that the system crashes.

One advantage of the SQL Server is that, every database has a transaction log and logging is enabled by default. The size of the transaction log is also determined dynamically by the server and log truncation automatically occurs after certain events (Microsoft Developer Network, 2012). Looking at the volume of information stored in the different log files and the system databases, it is possible that data can be reconstructed on an SQL Server during its forensic investigation provided that the structure of the logs are understood. However, it might be an added advantage in some cases, if the system administrator can specify the size of the transaction log.

PostgreSQL

PostgreSQL (or Postgres) is an open source object-relational database management system that was derived from the POSTGRES package written at the University of California at Berkeley (The PostgreSQL Global Development Group, 2013). Postgres uses two main types of logs[2] both of which are helpful for forensics analysis; the Write-ahead logging (WAL) or transaction log, and the Server log.

Write-ahead logging (WAL) is a standard method in Postgres used specifically for maintaining data integrity. The WAL log is used to store changes that occur in data files before the changes are made on the data. This allows the recovery of a database from the WAL log in the event of a failure by retrieving changes that have not been applied to the data pages from the WAL log records. WAL uses several configuration parameters that affect the volume of the information stored in the log, and as such the performance of the database (The PostgreSQL Global Development Group, 2013). One of the parameters of the WAL configuration is the *wal_level*. The *wal_level* parameter determines how much information is written to the WAL log and can be set either to *minimal*, *archive* or *hot_standby* at server startup. The *archive* setting allows the logging of information needed to recover from a crash and enables the archiving of the WAL logs. The *hot_standby* setting allows the storage of additional information that are required to reconstruct the status of running transactions in the WAL log and also allows logs to be archived. The *minimal* setting keeps only the information required to recover from a crash or immediate shutdown. It avoids the logging of bulk operations, in order to make the operations faster. The default *wal_level* setting on Postgres is the *minimal* setting. Unfortunately, the *minimal* setting of WAL does not contain enough information that can be useful for reconstructing data from the WAL logs or from the backups (The PostgreSQL Global Development Group, 2013).

The server log in Postgres is used mainly for error reporting, server messages, and logging of executed SQL statements. There are various parameters that determine what the database logs and when it logs in the server log. Some of the parameters that affect the volume of information that may be available in the server log for forensic investigations include the following (The PostgreSQL Global Development Group, 2013):

1. *log_min_messages*: this parameter controls the level of messages that are written to the Postgres server log. The possible settings are PANIC, FATAL, LOG, ERROR, WARNING, NOTICE, INFO, DEBUG1, DEBUG2, DEBUG3, DEBUG4, DEBUG5. Every level logs all the information contained in the preceding level together with other additional information. The default setting is NOTICE.
2. *log_min_error_statement*: this parameter determines which SQL statements that generate an error condition will be included in the server log. The possible settings are the same as for the *log_min_messages* parameter and the default setting is ERROR, which means that only statements causing errors, fatal errors or panics will be logged.
3. *log_statement*: this parameter allows the specification of which SQL statements are logged regardless of whether or not an error occurs. The possible setting include **none** (no statement is logged), **ddl** (data definition queries are logged) and **mod** (all data definition and data modification queries are logged). The default setting is **none**, which implies that it is impossible to find queries executed in a Postgres database uses a default log setting should a forensics analysis be required.

Oracle

The Oracle database is a relational database management system that includes some aspects of other paradigms such as hierarchical and object oriented models (Oracle). Although evidence can be collected from various sources in an Oracle database, for example, listener logs, alert logs, sqlnet logs, intelligent agent logs, and access logs (Wright, 2007), there are three major logs in Oracle that are of utmost importance for reconstruction in a forensic

[2] Other log files in PostgreSQL include the event log and the server log (The PostgreSQL Global Development Group, 2013).

analysis of the database; the redo logs, the archived redo logs and the alert logs.

The redo log in Oracle is the most important part of Oracle when it comes to recovery or forensics analysis. Every instance of an Oracle database has a redo log that serves as a protection of the database against failure. The log consists of two or more log files that are used to store all the changes made to the database as they occur. Records in the log files describe changes made to single blocks in the database. The data stored in the files can be used to identify all the changes made to the database as well as the undo segments (Oracle). The redo log is enabled by default on the Oracle database.

Groups of redo log files can be saved to offline locations by using the archived redo logs. The archived redo logs allow the recovery of a database in the event of a crash or disk failure, allows the updating of a standby database, and provides information about the history of a database when using the LogMiner utility in Oracle. There are two modes that can be activated to tell the database whether or not to archive the redo logs:

1. NOARCHIVELOG: this mode disables the archiving or redo logs. Although this mode protects a database from instance failures, it does not allow the recovery of a database in the event of a system failure.
2. ARCHIVELOG: this mode enables the archiving of the logs and can be done manually or automatically.

The archiving mode is usually selected at the time of creation a database and is by default set to NOARCHIVELOG.

Alert logs and trace files are used to keep record of errors that occur in an Oracle database. The alert log keeps a chronological record of messages and errors involving data definition queries, administrative operation statements, errors relating to functions of shared servers and processes, and errors with values of initialization parameters that are different from the default values. Background processes write to a trace file when an internal error is detected by the process. The information written to a trace file may be used by an administrator to tune applications and instances of the database. The default setting for trace files is the trace level 0, which disables archive log tracing.

DB2

IBM DB2 is a relational database management system that uses two main types of logs[3] in its operations; the database recovery log (or transaction log), and the diagnostic information log.

The database recovery log keeps record of all the changes made to the database, including created tables and updates on existing tables. It is useful for ensuring that the database can be recovered from a failure and that the database is in a consistent state (IBM, 2012). In the event of a failure, the changes already made to the database but which are not yet committed are rolled back while the committed transactions that have not been written to disk are written by using the information available in the recovery logs, in order to preserve the integrity of the database (IBM, 2012). The logging option can be set either as *circular logging* (which does not allocate more space for a log file when it becomes full, but overwrites the same file provided that the records in the file have been committed) or *infinite logging* (where full log files are not overwritten but are closed, and archived).

The diagnostic information log files consist of various log files that can be used to troubleshoot and diagnose errors and causes of errors in a DB2 database. The diagnostic information log files include dump files, trap files, administration notification log files and alert log files. These files are stored in a single directory and can be sorted or merged based on the date and time stamps included in the files. There are various parameters that can be used to control how much information is logged and how the log files are handled in DB2. Many of default settings are already the best choices for reconstruction. However, we highlight some of the parameters with default settings that may not be beneficial for reconstruction during a forensic investigation below:

1. *log_ddl_stmts*: this parameter is used to specify whether or not extra information regarding data definition statements should be written to the log. The default setting is NO, which implies that such statements are not logged.
2. *logarchmeth1*: this parameter is used to specify the log archiving method that will be used in the database. The possible settings include OFF and LOGRETAIN. The default setting is OFF which implies that logs are not archived and the database uses *circular logging*. Thus full logs are overwritten and the database is not roll-forward recoverable (IBM, 2012).
3. *logfilsiz*: this parameter is used to define the size of the log file. The possible range is from 4 to 1048572 (in kb) and the default is 1000.

Sybase

Sybase IQ (or Sybase) database is a relational database with a fundamental difference in that it focuses on the readers and not writers. That is, it is aimed at providing fast responses to queries from many users (Sybase, 2010). The two logs in the Sybase DBMS are the transaction log and the message log.

Sybase automatically handles the creation and deletion of its transaction log and the database server always requires a transaction log to run. A transaction log mirror (duplicate) is also maintained by the database. Although it is not required, the mirror provides an alternative source of information and may be useful in places where regular backups are not maintained (Sybase, 2010). The transaction log stores changes made to the database including, version information, free spaces and other information that may be useful for recovery and auditing of the database. Over time, the size of the transaction log may grow very large and it may be necessary to truncate the log. The method used to truncate the log is a decision made by the administrator,

3 In Linux, UNIX and Windows operating systems, DB2 also offers error logs that are specific to each platform.

but can be done either by stopping the database and making a copy of the log or by using the backup utility in Sybase.

A message log is created for every newly created database in Sybase. By default, Sybase logs error messages, status messages, insertion notification messages, and query plans in the message log file. The message log file grows to an unlimited size by the default setting and exists until the database is dropped. Archiving is not enabled by default in Sybase.

Summary of default log settings

From the survey of the information logged by default in the popular databases described in MySQL Section through Sybase Section, it is evident that the default logging configuration in many DBMSs does not generate logs containing the information often required for a database forensics analysis and reconstruction.

As an example, if no SQL statements with errors are executed on a Postgres database with the default configuration (which sets the *log_statement* parameter to **none**), the log file generated contains no information except the startup time of the database, as shown in Fig. 1. If any query executed leads to an error, the statement is included in the log file as shown in the excerpts in Fig. 2 since the *log_min_error_statement* and the *log_min_messages* parameters are set to allow logging of errors and the statement causing the errors by default, respectively. Although the inclusion of erroneous queries in the log file may be useful, the log contains no information that can be used for reconstruction if none of the statements executed generates an error.

Considering all the popular database discussed above in general, some of the issues with the default logging configuration are summarized below.

- Not all the available logs on a DBMS are usually enabled by default. For example, among the six DBMS considered in this paper, only one DBMS (SQL Server) has all its logs enabled by default.
- In cases where some logs are enabled, the most valuable log (in terms of data used for forensic analysis) is sometimes excluded in the default configuration. For example, only the error log is enabled by default in MySQL, even though the other logs may contain more valuable data for reconstruct if they were enabled.
- The level of logging selected by default on many DBMS allows the storage of only a minimal amount of data specifically for crash recovery, or the log size is set to a default value as is the case with the SQL Server, Postgres and DB2 database systems.
- Although many DBMS support the archiving of logs or the use of an online storage, the default logging preference on most systems does not support the archiving of logs and the logs are usually overwritten.

These issues reflect the need to specify the ideal log settings or the log data that are required for database logs to be effective for analysis and reconstruction in database forensics.

Ideal log settings

As mentioned earlier, the concept of ideal log setting refers to the combination of logging preferences on a database that enables an effective reconstruction during a forensic analysis of a database. In this section, we identify the information that is required in a database log file in order for it to be useful for reconstruction during a forensic analysis of the database. A general definition of the required set of information is given since the content of the log files in different databases differ and the settings on different databases are unique in most cases. The set of required information may be regarded as the ideal log requirements for reconstruction in database forensics. We also consider the identified requirements in relation to the operations of the databases discussed in Default Log Section and determine the log settings and preferences that should be enabled in the databases in order to satisfy the log requirements. It is important to note that the definition of an ideal log requirement does not imply that database log files that do not meet the requirements should not be included in the data collected for a forensic analysis. Although it may be more difficult to reconstruct information using log files that do not store all the information identified, there might be additional techniques that may be employed for forensics analysis in specific databases. Also, since an investigation can be influenced by other factors such as organizational policies, regulations or technical limitations, an ideal may be subject to such factors. The focus of this section is specifically on the ideal log for reconstruction purposes.

Definition of an ideal log requirement for database forensics reconstruction

In many databases, there is a trade-off between the performance of the database and the volume of information that can be stored in log files used for recovery or for ensuring the consistency of the database in the event of a failure or other incidents. Unfortunately, the log files are often the richest source of information during the forensics analysis of a database. Ensuring that adequate logs are available on a DBMS may also serve as part of a forensic readiness plan (Barske et al., 2010; Rowlingson, 2004) for future incidents that may involve the database. And as such, it is important to ensure that vital information that

```
2014-10-07 08:59:21 CAT FATAL:  the database system is starting up
2014-10-07 08:59:22 CAT LOG:  redo is not required
2014-10-07 08:59:22 CAT LOG:  database system is ready to accept connections
2014-10-07 08:59:22 CAT LOG:  autovacuum launcher started
```

Fig. 1. Log File with Default Parameter *log_statement* = *none*.

```
2014-10-07 10:03:38 CAT ERROR:  table "persons" does not exist
2014-10-07 10:03:38 CAT STATEMENT:  drop table persons;

2014-10-07 10:09:12 CAT ERROR:  syntax error at or near "Alter" at character 559
2014-10-07 10:09:12 CAT STATEMENT:  Alter table Persons rename column City Town;
```

Fig. 2. Log File showing Query with Error.

can be used during the forensic analysis of a database exists in its log files, particularly in the transaction logs (or its equivalent) of the database. Other log files, such as the error logs and server logs should also contain as much information as possible that may be useful in the event of an incident as much as possible.

The required information that should be stored in a database log can be determined by considering the objective of a database forensic analysis. According to the frameworks for digital forensic investigation process that have been developed by various researchers (Beebe and Clark, 2005; Carrier and Spafford, 2003; Ieong Ricci, 2006), the objective of the reconstruction phase of a digital forensic analysis is to draw conclusions from various pieces of information and provide answers to the what?, who?, when?, where?, why? and how? questions. Ieong (Ieong Ricci, 2006) describes the FORZA framework which gives a technical overview of various roles in an investigation and how they are interrelated through these questions. In order to identify the set of information required for database forensics reconstruction, we also consider these questions and highlight four aspects that are usually of interest in database forensic analysis based on the questions. These aspects are described below:

- Identifying changes made to a database: this aspect corresponds to the what? question of the reconstruction process and involves the recognition of the information required to achieve this objective.
- Identifying who may be responsible for the changes: this aspect of database forensics deals with the who? and where? questions together with the recognition of the information necessary to achieve this objective.
- Confirming what we expect to see in the database: this aspect corresponds to the why? and how? questions of the reconstruction process. Information about why a database (or piece of information) is the way it is on the database and how this might have happened assists in confirming the information in the database.
- Determining the timeline of events on the database: the when? question of the reconstruction process is handled in this category. Information about when an event occurred in a database is required to determine a timeline of events that occurred in the database.

As mentioned earlier, much of the previous work in describing the process of reconstruction or the forensics analysis of a database assume that the required information is available in the log files. Unfortunately, this is not always the case. In Default Log Section, we have discussed some of the popular DBMS and highlighted various default settings that may not enable the storage of the required information on a database. In order to determine the logging preferences or settings that would allow an effective database forensics reconstruction to take place in a database, we identify the information that should be stored in a database log by considering each of the aspects described above.

Identifying changes made to a database

Identifying the changes made to a database is arguably the most important aspect of a database forensic analysis. The reconstruction phase of database forensics relies mainly on the ability to identify what was done to the database. In order to identify changes to a database, the database log should contain the information below.

Data modification queries: The fact that many of the work that has been done on the forensic analysis of a database (Litchfield, March 2007a, 2007b; Frühwirt et al., 2010, 2012; Fowler, 2008; Fasan & Olivier, 2012a, 2012c) rely on the log of queries that was executed on a database reflects its importance in database forensics. Since a database is typically usually modified by executing a query, information about queries that modify the data in a database should always be logged. While it is true that quite a number of databases enable the transaction logs by default, the volume of data stored is often minimal with the default setting. Logs of data modification queries should contain all the information needed to be able to reconstruct such queries. This is particularly useful for an investigator when there are no backups or when a value of interest occurred in between two backups during an investigation. In such a case, the inverse of the queries may be found in order to reconstruct data in these cases (Fasan & Olivier, 2012a, 2012c). The information needed to maintain the consistency of a database and perform recovery should also be stored in the appropriate log files on a database.

Data definition queries: Unlike data modification queries, most databases do not keep record of data definition queries by default. This information is of utmost importance when reconstructing data because it gives an insight into what tables should or should not be present on a database and why. This functionality should be enabled by an administrator and information about the time of execution, the user who issued the query, and other details of data definition queries such as CREATE, ALTER, DROP, RENAME, TRUNCATE should be included in a log file. Details of unsuccessful queries in this case should also be included in the error logs.

This information is also required in confirming the consistency of the information on the database with what is expected to be on the database.

Metadata changes: As mentioned earlier, queries executed on a database may give a false result due to changes made to the metadata, making it difficult for an investigator to comprehend some of the changes that might have been made to a database. Since the output from a database is dependent on the data stored in it and the metadata describing the data, it is possible to get incorrect information from queries if the metadata has been tampered with. As such, the changes made to a database metadata or other database files should be logged regardless of whether or not the changes were carried out by the administrator, a program or a malicious user. The logging of metadata changes will enable an investigator to get information about the previous state of the metadata, especially where a database rootkit is involved. In a paper by Beyers et al. (2011), the authors described techniques for obtaining a clean investigation environment of a database where one or more layers of the database metadata have been compromised. The logging of metadata changes in a database log file will also be a useful approach in identifying changes and obtaining a clean (or uncompromised) version of the metadata during a database forensics analysis.

Identifying who may be responsible for changes

Another aspect of database forensics analysis involves an attempt to identify who may be responsible for the changes made to a database. In order to achieve this objective, it is important that the following information are available in a database log.

Access information: In order to identify the users that are logged on or that may be responsible for an event that occur within a particular time frame, details of user logins and logouts of a database should be logged in the server log file (or its equivalent) of a database. Analyzing previous logins and logouts of a database may reflect certain patterns and assist an investigator in identifying anomalies in such patterns. The identification of anomalies can be useful in reconstructing event that occurred in a database or even the data that was stored in it.

Data access queries: Although database access information may show the users that are logged in on a database at some time, it does not reflect who might have accessed a particular information. As discussed in Default Log Section, the default logging preference on most databases do not enable the logging of accesses to data on the database. This is an important information that may be useful in identifying intents when investigation an incident and as such should be logged. It may also be useful in recognizing when a data was last accessed and who accessed it. Log records about data accesses should include at least the name of the table that was accessed, the user that issued the query, and the date and time stamp of when the query was performed. Details of unsuccessful accesses should be included in the error logs.

Changes in privileges: To understand log records about data access queries, it may necessary to know more the privileges given to a certain user. Information regarding changes in the privilege given to anyone with access to a database should be logged with the date and time of such changes. This enables an administrator to occasionally review privileges given to users and assist an investigator to identify possible culprits in the event of certain crimes, such as data theft.

Database administrator operations: Since the database administrator has access to all the information on a database, it is necessary to know the operations performed on the database by the administrator. Information about operations performed on a database by an administrator or anyone with similar access privilege should be logged, especially in cases where the operation may have a "critical" effect on the database. The definition of what is "critical" should be known to the database administrator and the database. This may be useful in identifying the operations performed by anyone or anything pretending to be the administrator when the database has been compromised.

Confirming the information in the database

This aspect of database forensics analysis deals with checking the consistency of the database with what the investigator or administrator expects to see on the database logs. Some of the information that should be available in the logs for this purpose are described below.

Server information: In order to confirm the information in a database, it is important that the state of the database can be determined. The log should contain information that will be useful in understanding failures, identifying why a particular log may be missing certain information, and any other issue that may arise during a forensics investigation of the database. Thus, events that occur during database start-ups, stops or restarts should be logged in a database log file. This information should reflect the conditions or configurations under which a database was started or restarted, as well as information about how the database was last stopped. Errors and messages displayed during these stages should also be logged. Operational messages such as errors that occur during a backup operation or log archiving process (when enabled) should also be included in one of the database log files.

System failures and errors: Errors logs are the most popular log files that are maintained on almost every DBMS. Error logs can sometimes reflect abnormal conditions in data and show evidence of tampering. It can also be used for time analysis during an investigation. However, the volume of information stored in the error logs differs depending on what the database has been configured to log. Error logs should consist of details about system failures and other failed internal processes. This information may become useful in knowing the evidence to look for and where to look when conducting a forensics analysis of the database. An error should contain important pieces of information about each error, such as, the cause, the date and time of occurrence, and any failed attempt by the DBMS to correct the error.

Relocation of log files: Many databases contain a default directory used for storing log files but also allow a different directory to be specified. Records of events that causing the relocation of log files will be useful for reconstruction and forensic analysis in cases where a malicious user has copied files from their original location to other places (or deleted them) to hide their activities. Therefore, events that cause changes in the default directory for storing log files, or the copying and/or deletion of log files should be logged in one of the database log files. This type of logging should also apply to the relocation of files that are expected to be in a particular location in a database.

Determining the timeline of events

As with digital forensics in general, creating timeline of events can assist an investigator to gain insight into the events that occurred and the people involved. It also assists in identifying patterns and anomalies that may reveal other sources of evidence in a system. All the ideal log information described earlier are useful in the creation of a timeline of events. An additional information that is required for the creation of a timeline of events is the archive of all the information identified above.

Archived logs: In many databases, the size of the log files is kept to a certain maximum by overwriting the file from the top when it becomes full (for example, with circular logging in Oracle). Archive of database log files should be maintained rather than allowing overwrites. Archives are very useful in reconstructing data since a single log file may not contain all the information of interest due to overwrites. Log archives provide a rich source of information for an investigator regardless of how long ago the data of interest might have existed or have been manipulated on the database.

Challenges and solution for ideal logging preferences

Logging of all the information prescribed above doubtlessly creates various challenges that affect the operations and the investigative process of a database. Some of the challenges that may be faced when logging all of the required information are highlighted below.

- Effect on database performance: in most databases, logging causes an overhead that affects query performance. Although error logs and server logs (or their equivalents) have little effect on a database, transaction logs and audit logs can significantly reduce the speed of a database in completing queries due to the number of records that may need to be written to such logs before it is completed or updated on the database. Despite this, the increasing need to ensure data security and provide a means of reconstructing data requires that adequate log records are kept at all times.
- Volume of information in log files: One of the challenges of handling digital evidence is the volume of information that can be retrieved during an investigation. Logging of all the information described above implies that the same problem is encountered in database forensics. Tools and techniques that can be used to extract useful and relevant information from the logs will be required in order for the logging process to be worthwhile. However, even in the absence of analysis tools, database log records are very useful for forensics analysis, even though more effort will be required for a manual analysis.
- Log file formats and contents: Another challenge with handling database log files is the various file formats used in different databases. In addition, some databases also provide the option of logging into tables on the database itself. Thus, it is required that an investigator understands the format used by the database being investigated and how to handle the data in such format. Also, data records stored in the log files of different DBMSs differ both in organization and in length. For example, certain information about a similar query may be logged in one DBMS and not logged in another even when similar settings are used. This makes the automation of log analysis an issue that has to be handled with a specific DBMS in mind.

Some of the techniques that can be employed to handle these challenges or minimize their effects on database forensics analysis are described below.

- Storing log files on separate physical mediums: a possible way of handling the volume of log files and its effect on database performance is to store archived log files on a different physical medium from the database itself. This is beneficial particularly in systems that use different data access formats for the logs and the data files. For example, in Sybase, the catalog and the IQ store (which contains the data files and data dictionary) are randomly accessed while the transaction log is sequentially accessed. An additional advantage of this technique is that it also preserves the logs against loss or attack on the database and provides a way of retrieving the logs for analysis without affecting the operation of the database. A relatively similar alternative to this approach is to take regular backups of the log files before it is overwritten by default.
- Use of log analysis and/or log management tools: to enhance the analysis of the volume of information that may be retrieved from log files during database forensics, the analysis process should be automated. Many databases provide tools that can be used for analyzing the log files. For example, the Log Analysis tool in DB2 (IBM), the Oracle LogMiner (Wright, 2005), and the Microsoft SQL Server Profiler (Rankins et al., 2010). Very few external tools are also available for the analysis of specific databases, for example, the pgFouine[4] tool that can be used for analyzing Postgres logs. These tools provide an efficient way of analyzing logs, even though they can only be used in the specific DBMS for which they were designed. Tools and techniques used in the general field of digital forensics may also be useful for analyzing logs in specific formats in some cases. Chuvakin (2007) discussed the need for a more advanced database log management tool that is capable of working across various DBMSs and automating not only the process of log analysis, but also the process of collection, transferring and storing of log files. The availability of an advanced log management tool will also alleviate the problem of handling and combining logs in tables and in files. It is also important that such tools have the capability to reconstruct data based on the information in the log files.
- Occasional review of logs and auditing: an occasional review of logs is required in order to provide insight about what information should be excluded from the

[4] http://pgfouine.projects.pgfoundry.org/.

logs or needs to be included. This may assist in reducing the volume of irrelevant data in logs during an investigation. In addition, the auditing feature (when available) of a database should be enabled as it offers a level of protection for log files and other database files, and enhances the early detection of suspicious activities on a database.

Identification of possible ideal logging preferences

In this section, we identify the possible logging preferences that may be used to ensure that the required information is present in the log files of the popular databases considered in Default Log Section. We note that these preferences are meant to explain possible settings that can be enabled (which are not enabled by default) and do not imply that the settings are the only ideal settings. A definition of which information (amongst the log requirements discussed above) is of utmost importance in a specific database application will be required in realizing the ideal logging preferences in any particular application. The possible settings for the databases are considered below.

MySQL

MySQL server consists of various logging options (all disabled by default) that can provide a lot of information. To explore the capability of these logs for forensic analysis, the logs should be enabled. The slow query log may be viewed as a subset of the general query log and may be excluded. However, the General query log, the Binary log and the Relay log should be enabled on a MySQL server. The server automatically deletes relay logs once they are no longer needed, so the space required for relay logs and their effect performance is minimal. The size of the general query log and the binary log may grow large over time; some of the techniques described earlier (Challenges and Solution for Ideal Logging Preferences Section) may be employed to combat the space requirement problem.

Microsoft SQL server

As mentioned earlier, logging is enabled by default on SQL server. However, a default size is specified for the log file. It may be necessary to increase the size of the log, especially the transaction log in some cases. The *autoshrink* option, which allows the shrinking of specified files on the database, is also set to false by default. While it is not advisable to shrink data files on a database, the log files can be configured to shrink when they become big. To handle the problem of transaction logs growing big over time, the use of backups or a different physical medium for logs can be explored to ensure that performance is not significantly hampered.

Postgres

In order to exploit the potential capability of the WAL logs in reconstructing the information on a Postgres database, the *wal_level* setting for the WAL log should be set either to the *archive* or the *hot_standby* setting before starting up the database. In addition, the *log_min_messages*, *log_min_error_statement* and the *log_statement* parameters of Postgres should be given values that allow the logging of enough information about queries. For example, the *log_statement* parameter should be set to **mod** so that all data definition and data modification queries can be logged.

In contrast to the example given in Section Summary of Default Log Settings, if the *log_statement* parameter is set to **ddl** during the execution of the statements in Fig. 3, the log file generated contains all the data definition queries but no data modification queries together with a time stamp of when the query was executed as shown in Fig. 4. If the parameter is set to **mod**, the log contains a record of all data definition and data modification queries as shown in Fig. 5. Setting the parameter to **all** causes every statement executed on the database, including select statements to be logged.

Another example on the opposite end would be to set the *log_min_messages* parameter to LOG or other lower (in terms of information stored) options. The effect of this is that queries with errors are no longer logged. Some information about the status of the database are also excluded from the log file in this case. For example, setting the *log_min_messages* parameter to LOG causes changes to the database configuration file to be recorded together with information about what was changed in the configuration file, whereas setting the parameter to a lower option (say FATAL) does not log what is changed in the configuration file even though it records that the configuration file was reloaded.

```
CREATE TABLE Persons(
P_Id int NOT NULL PRIMARY KEY,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255));
INSERT INTO Persons (P_Id, LastName,firstName, Address, City)
VALUES (12345, 'Benson','Pritchard B.','Watermeyer','Pretoria');
UPDATE Persons SET City = 'Johannesburg' WHERE City = 'Pretoria';
SELECT * FROM Persons;
ALTER TABLE Persons RENAME COLUMN City TO Town;
DROP TABLE Persons;
```

Fig. 3. Queries executed on database.

```
2014-10-07 13:29:54 CAT LOG:  statement: CREATE TABLE Persons(
P_Id int NOT NULL PRIMARY KEY,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255));
2014-10-07 13:30:59 CAT LOG:  statement: ALTER TABLE Persons RENAME COLUMN City TO Town;
2014-10-07 13:31:48 CAT LOG:  statement: DROP TABLE Persons;
```

Fig. 4. Log file with parameter *log_statement* = *ddl*.

Oracle

The ARCHIVELOG setting in Oracle should be enabled to exploit the rich source of information that archived logs provide. The auditing capability of the database should also be turned on. The default archive logs trace level of 0 (which disables archive log tracing) should also be changed to 1 (by setting the parameter LOG_ARCHIVE_TRACE = 1) so that information or errors about the process of archiving log files can be retrieved.

DB2

In order to be able to reconstruct data definition statements, the *log_ddl_stmts* parameter in DB2 should be changed from the default NO value to YES. The *logarchmeth1* parameter which is used to specify the log archiving method is also set to OFF by default. This makes it impossible to access earlier log records as they are overwritten when using this default setting. A better alternative value for this parameter is the LOGRETAIN setting, which specifies that log files should be retained and become online archived log files which can be subsequently used for roll-forward recovery (IBM, 2012). If performance is an issue, and the LOGRETAIN option cannot be used, the default size of the log should be increased as much as possible in order to ensure that enough information can be retrieved from the logs when required. The use of a different physical medium for log archives can also be explored.

Sybase

Using the default settings in Sybase, message logging stops if the disk becomes full while writing a message to the log and a record of this is made in the server log. Although message logging also resumes by default when the error condition is resolved (by creating a new log file), it would be better to avoid log files growing indefinitely and instead archive relatively shorter log files. This makes it easier to recognize relevant log files and is also beneficial in the event of a system crash or disk failure. In addition, rather than stopping the database to truncate the transaction log, the backup facility in Sybase should be employed. This allows the availability of more information in the event that a live forensics (Carrier, 2006) of the database needs to be conducted.

Database forensics reconstruction dimensions and log files

In this section, we consider each of the required set of information identified in Section Definition of an Ideal Log Requirement for Database Forensics Reconstruction in relation to the three dimensions of reconstruction in database forensics (Fasan and Olivier, 2012b). This is to ensure that the required information is applicable in the various dimensions and that additional requirements for any of the information are identified in each dimension.

Modified databases and the ideal log setting

Amongst the three dimensions of database forensics reconstruction, the modified database category is able to provide the most readily usable log files, since the database (and therefore, the log files) have not be compromised or damaged, and requires no additional steps in order to obtain the log files. As such, the required set of information described in Section Definition of an Ideal Log Requirement for Database Forensics Reconstruction completely applies to a modified database. One of the advantages of storing these required set of information in the log of a modified database is that the information in the log may reflect inconsistencies that expose a compromise or damage to the database during an investigation, such that a database initially considered to be a modified database may eventually have to be investigated as a compromised and/or a damaged database. When the objective of the forensics analysis of a modified database is to retrieve data that

```
2014-10-07 13:54:13 CAT LOG:  statement: CREATE TABLE Persons(
P_Id int NOT NULL PRIMARY KEY,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255));
2014-10-07 13:58:12 CAT LOG:  statement: INSERT INTO Persons (P_Id, LastName,firstName, Address, City)
VALUES (12345, ’Benson’,’Pritchard B.’,’Watermeyer’,’Pretoria’);
2014-10-07 13:59:15 CAT LOG:  statement: UPDATE Persons SET City = ’Johannesburg’ WHERE City = ’Pretoria’;
2014-10-07 14:01:55 CAT LOG:  statement: ALTER TABLE Persons RENAME COLUMN City TO Town;
2014-10-07 13:02:13 CAT LOG:  statement: DROP TABLE Persons;
```

Fig. 5. Log file with parameter *log_statement* = *mod*.

existed as some earlier time, and an archive of logs is available, it may be necessary to identify and analyze only the relevant logs, so as to reduce the volume of information that needs to be processed.

Compromised database and the ideal log setting

The main concern during the investigation of a compromised database is that the information provided by the database cannot be trusted. This concern may also affect the use of some or all of the information in the log files on the compromised database. In order to ensure that the data in the logs have not been compromised, it is important that an investigator checks the consistency (or inconsistencies) of the logs and also the hypotheses made based on the log records (Cohen, 2011). To check that a log is consistent, the investigator may need to confirm that the log reflects only the information that was selected to be logged in the configuration, and that the data format of the files are as expected. On the other hand, checking the consistency of hypotheses made may involve an attempt to regenerate the current instance of a database from a reconstructed version of the database. If the regenerated instance differs from the actual database instance, then it shows that some information might not have been logged or there might have been some compromise of the database. This process may also be useful in identifying the events or steps taken by an attacker when investigating an incident.

On the other hand, there is a large amount of data to support the investigation of a compromised database if the required set of information described in Section Definition of an Ideal Log Requirement for Database Forensics Reconstruction are logged. For example, if changes to a database metadata are logged, then it is possible for an investigator to identify the changes made to compromise a database and when the changes were made during an analysis. Also, the logging of data definition queries is useful for identifying changes made to the tables on a database. Other information in the log also provide bits and pieces of data that can be used by an investigator.

Damaged/destroyed databases and the ideal log setting

One of the challenges of investigating a damaged or destroyed database is the fact that the data files or log files of the database may have been modified, deleted or moved into other locations different from the expected location. Similar to the use of log files in the investigation of a compromised database, it is important for an investigator to check for any inconsistencies in the hypotheses made based on the log records (Cohen, 2011) in cases where the database is still operational. Differences in the reconstructed database instance and the actual instance of the database may be useful in identifying missing information in the log file or sections of the log file that have been modified. In cases where the database is no longer operational or where the log file has been deleted, file carving techniques (Carrier, 2005) may be used in regenerating the log file (or other files) of the database for forensics purposes.

Some of the advantages of having the required set of information described in Section Definition of an Ideal Log Requirement for Database Forensics Reconstruction during the investigation of a damaged database include the fact that if the information about the relocation of log files are stored (Section Confirming the Information in the Database), then the log file can be easily located when it is moved to a different location. Information about events relating to database startups, stops and restarts (server information) is also a good pointer to the possible intents of a database attacker. Access information and metadata changes recorded in the log file serves as a valuable source of information for identifying what was changed and by who during database forensics. The log of changes in privileges allows an investigator to identify privileges that should not have been given, and this may be useful in identifying the intent of possible culprits. It is possible that the damage to a database was carried out by an attacker pretending to be an administrator, the log of database administrator operations will be particularly useful in this case.

In summary, the required set of information for database forensics described in Section Definition of an Ideal Log Requirement for Database Forensics Reconstruction applies to all the dimensions of database forensics. Although additional steps may be required to obtain or use the log files from compromised and/or damages databases, the information ensures that as much information as possible is available to assist an investigator in carrying out the forensics analysis of a database and reconstructing data.

Conclusion and future work

The aim of this paper is to expose the inadequacy of the default logging preferences on most databases in obtaining log files with as much information as required for an effective reconstruction during database forensics and to specify the required information for an effective reconstruction process. The paper describes the default logging preferences of six of the popular database management systems and points out specific options and parameters that affects the volume and relevance of the information that will be logged. The paper also determines the set of required information that should be present in a log file for it to be significantly useful for reconstruction. The challenges involved with storing the required information as well as some of the tools and techniques that can be employed to overcome the challenges are also discussed. In addition, the possible logging preferences that will enhance reconstruction when investigating the popular databases mentioned are identified. To make matters more concrete, we focus on the various dimensions of database forensics reconstruction and identify the usability of the identified required information in the different dimensions. Additional requirements and/or the techniques required for log analysis and usability in the different dimensions are also identified.

Future work will include reviewing the identified set of required information in different DBMSs and application areas to understand if or when some information may be excluded in certain cases. An assessment of available log management tools and their usage in handling database log files will also be necessary in order to discover any additional requirements for such tools. In addition, work is still required on the comparison of regenerated database instances with an actual database instance, and the use of their consistencies or inconsistencies in supporting the forensics analysis of a database.

Although the ideas discussed in this paper may be useful in terms of forensic readiness, further research is also still required on the forensic readiness of database management systems since there are factors apart from the logs (and the ability to reconstruct data) that need to be considered in this regard and this is left as future research.

Acknowledgment

This research was supported by the Organization for Women in Science for the Developing World (OWSD).

References

Adedayo Oluwasola Mary, Olivier Martin S. On the completeness of reconstructed data for database forensics. In: Digital forensics and cyber crime. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol. 114. Springer Berlin Heidelberg; 2013. p. 220–38.

Barske D, Stander A, Jordaan J. A digital forensic readiness framework for South African SME's. In: Information Security for South Africa (ISSA); 2010.

Beebe Nicole Lang, Clark Jan Guynes. A hierarchical, objectives-based framework for the digital investigations process. Digit Investig 2005;2(2):147–67.

Bevel T, Gardner RM. Bloodstain pattern analysis: with an introduction to crime scene reconstruction. 2nd ed. FL: CRC Press; 2002.

Beyers Hector, Olivier Martin, Hancke Gerhard. Assembling metadata for database forensics. In: Advances in digital forensics VII. IFIP Advances in Information and Communication Technology, vol. 383. Springer Berlin Heidelberg; 2011. p. 89–99.

Carrier Brian, Spafford Eugene H. Getting physical with the digital investigation process. Int J Digital Evid 2003;2(2):1–20.

Carrier Brian. File system forensic analysis. Upper Saddle River, NJ: Addison-Wesley Professional; 2005.

Carrier Brian D. Risks of live digital forensic analysis. Commun ACM 2006; 49(2).

Carrier Brian D. A hypothesis-based approach to digital forensic investigation. West Lafayette, Indiana: Purdue University; 2006. PhD thesis.

Casey Eoghan. Digital evidence and computer crime forensic science. Computers and the Internet. 3rd ed. Academic Press; 2011.

Chuvakin Anton. Introduction to database log management. Technical report, LogLogic Inc. 2007., http://www.infosecwriters.com/text_resources/pdf/AChuvakin_DB_Logging.pdf [accessed 19.03.13].

Cohen Fred. Digital forensic evidence examination. 3rd ed. Fred Cohen & Associates; 2011.

Elmasri Ramez. Fundamentals of database systems. 5th ed. Pearson Education; 2008.

Fang Ru, Hsiao Hui-I, He Bin, Mohan C, Wang Yun. High performance database logging using storage class memory. In: IEEE 27th International Conference on Data Engineering (ICDE); 2011. p. 1221–31.

Fasan Oluwasola Mary, Olivier Martin S. Correctness proof for database reconstruction algorithm. Digit Investig 2012a;9(2):138–50.

Fasan Oluwasola Mary, Olivier Martin S. On dimensions of reconstruction in database forensics. In: Proceedings of the 7th International Workshop on Digital Forensics and Incident Analysis (WDFIA 2012). Plymouth University; 2012b. p. 97–106.

Fasan Oluwasola Mary, Olivier Martin S. Reconstruction in database forensics. In: Advances in digital forensics VIII. IFIP Advances in Information and Communication Technology, vol. 383. Springer Berlin Heidelberg; 2012c. p. 273–87.

Fowler Kevvie. SQL server forensic analysis. NJ: Addison Wesley Professional; 2008.

Frühwirt P, Huber M, Mulazzan M, Weippl ER. Innodb database forensics. In: 24th IEEE International Conference on Advanced Information Networking and Applications (AINA); 2010. p. 1028–36.

Frühwirt P, Kieseberg P, Schrittwieser S, Huber M, Weippl E. Innodb database forensics: reconstructing data manipulation queries from redo logs. In: Seventh International Conference on Availability, Reliability and Security (ARES); 2012. p. 625–33.

Garfinkel Simson L. Digital forensics research: the next 10 years. Digit Investig August 2010;7.

Gladyshev Pavel, Patel Ahmed. Finite state machine approach to digital event reconstruction. Digit Investig 2004;1(2):130–49.

IBM. IBM DB2 10.1 for Linux, Unix and Windows: database administration concepts and configuration. IBM; 2012.

IBM. IBM DB2 log analysis tool for z/OS, User's guide. 2nd ed. IBM; 2013. version 3 release http://pic.dhe.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.db2tools.ala33.doc.ug/alaugc30.pdf?noframes=true [accessed 19.08.13].

Litchfield D. Oracle forensics part 1: dissecting the redo logs. NGSSoftware Insight Security Research (NISR) Publication; 2007show March .

Litchfield D. Oracle forensics part 2: locating dropped objects. NGSSoftware Insight Security Research (NISR) Publication; 2007show March .

Litchfield D. Oracle forensics part 3: Isolating evidence of attacks against the authentication mechanism. NGSSoftware Insight Security Research (NISR) Publication; 2007show March .

Litchfield D. Oracle forensics part 4: live response. NGSSoftware Insight Security Research (NISR) Publication; April 2007.

Litchfield D. Oracle forensics part 5: finding evidence of data theft in the absence of auditing. NGSSoftware Insight Security Research (NISR) Publication; 2007show August .

Litchfield D. Oracle forensics part 6: examining undo segments, flashback and the oracle recycle bin. NGSSoftware Insight Security Research (NISR) Publication; 2007show August .

Litchfield D. Oracle forensics part 7: using the oracle system change number in forensic investigations. NGSSoftware Insight Security Research (NISR) Publication; November 2008.

Microsoft Developer Network. SQL server 2012. 2012. http://www.msdn.microsoft.com/library [accessed 04.04.13].

Olivier Martin S. On metadata context in database forensics. Digit Investig 2009;5(3–4):115–23.

Oracle. MySQL 5.6 reference manual. oracle, 2013. 2013., http://dev.mysql.com/doc/refman/5.6/en/server-logs.html [accessed 19.03.13].

Oracle. Oracle database administrator guide 11g release 1(11.1). 2013. http://docs.oracle.com/cd/B28359_01/server.111/b28310/toc.htm [accessed 19.03.13].

Otey Michael. SQL server log files. SQL Server Pro Community. 2006., http://sqlmag.com/sql-server/sql-server-log-files [accessed 19.08.13].

Pollitt Mark. A history of digital forensics. In: Advances in digital forensics VI, IFIP Advances in information and communication technology. Springer Berlin Heidelberg; 2010. p. 3–15.

Rankins Ray, Bertucci Paul, Gallelli Chris, Silverstein Alex T. Microsoft SQL server 2008 R2 Unleashed. Sams. 2010.

Ieong Ricci SC. FORZA - digital forensics investigation framework that incorporate legal issues. Digit Investig 2006;3:29–36.

Rowlingson Robert. Abstract a ten step process for forensic readiness. Int J Digital Evid 2004;2(3).

Stahlberg Patrick, Miklau Gerome, Levine Brian Neil. Threats to privacy in the forensic analysis of database systems. SIGMOD '07. In: Proceedings of the 2007 ACM SIGMOD international conference on Management of data. New York, NY, USA: ACM; 2007. p. 91–102.

Sybase. SybaseIQ 15.1.2 documentation. Sybase. 2010. http://infocenter.sybase.com/help/index.jsp [accessed 19.03.13].

The PostgreSQL Global Development Group. Postgresql 9.2.3 documentation. 2013. http://www.postgresql.org/docs/9.2/static/ [accessed 19.03.13].

Turvey BE. Criminal profiling: an introduction to behavioral evidence analysis. Academic Press; 2002.

Wright PM, Burleson DK. Oracle forensics: oracle security best practices. Rampant Techpress; 2010.

Wright Paul M. Oracle forensics in a nutshell. 2007. http://www.databasesecurity.com/dbsec/OracleForensicsInANutshell.pdf [accessed 19.03.13].

Wright PM. Oracle database forensics using LogMiner. 2005. Next Generation Security Software.