Database Application Schema Forensics

Hector Q. Beyers*, Martin S. Olivier†, Gerhard P. Hancke*

† Department of Computer Science, University of Pretoria
* Department of Electrical, Electronic and Computer Engineering, University of Pretoria

ABSTRACT

The application schema layer of a Database Management System (DBMS) can be modified to produce results that do not reflect the data actually stored in the database. For example, table structures may be corrupted by changing the metadata of a database, or operators of the database can be altered to produce incorrect results when used in queries. Such incorrect results may lead to a forensic examination to determine the cause of the problem. Alternatively, such modifications may be employed as an anti-forensic technique in an attempt to hide the actual data from an investigator when an investigation lead to the examination of a database. In both cases forensic examiners need to be aware of the impact of such metadata on queries and plan their examination of the database accordingly. Different versions of a layer of metadata may exist: a version as found on the computer being investigated, the version that was initially designed, versions from backups, and so on. It is possible that these versions are identical, but subtle ad hoc changes are often made over time and someone with access and malicious intent can introduce changes to modify the behaviour of the DBMS to achieve some nefarious goal.

This paper initially discusses categories of possibilities that exist to (surreptitiously) change the application schema; practical examples are used to illustrate these possibilities.

The paper is based on the premise that a specific combination of DBMS layers of metadata and data should be assembled to test specific hypotheses. For example, questions about how a DBMS should have responded to a specific query and how it does, in fact, respond are both facts that may be important to a forensic investigator. The paper illustrates how such a combination of layers may be of use to examine a specific facet of the behaviour of the DBMS. The paper refers to such a combination of layers as a configuration.

The primary purpose of the paper is to explore methods that may be used to construct a given configuration for testing. A process is proposed on how forensic evidence should be extracted from the application schema layer of a DBMS.

KEYWORDS: Database Forensics, Database Forensic Process, Database Abstract Layers, Application Schema Forensics

CATEGORIES: H.1.m, H.2.7

1 INTRODUCTION

In a period where the field of digital forensics attracted increasing interest, database forensics remains a relatively unexplored domain [1]. Database forensic research still frequently reflects on the reasons why this is the case; in fact, only a few years ago hardly *any* scientific research existed about database forensics despite the realisation that such work was urgently needed [2]. One possible reason for the lack of research is the inherent complexity of a database management system (DBMS) when compared to, say, file systems. While files are often abstracted as streams of bytes, a database is a collection of data where data elements are related to one another. Various layers of metadata influence the way in which other metadata (or data) is interpreted. A database table may, for example, not be stored in a manner that would be recognised as a table upon a cursory inspection. Metadata may exist that provides details that enables a user to 'see' the table. Such metadata may, for example, provide the name of the table, the layout (in terms of number of columns and the types of those columns), the names of columns, access rights that may hide the existence of some columns from some users, constraints that are placed on columns (or attributes of a row in that table), and so on. The table only takes shape when the bits that constitute the table are viewed through the 'lens' provided by the metadata. However, to understand this metadata one has to similarly make sense of it by looking at it through another metadata 'lens'. In principle this may lead to the need for an infinite chain of such 'lenses'. However, in a beautiful example of self-description this problem may be solved: if the metadata that describes a table can itself be stored in one or more tables, those tables describe themselves, thus obviating the need for an infinite chain. Clearly such a construction tends to be complex and subtle changes to any metadata may have far-reaching consequences on what is represented. Directly looking at the bits and bytes that constitute a database may cause the observer to misinterpret the semantics of what is

Email: Hector Q. Beyers `hqbeyers@gmail.com`, Martin S. Olivier `ms.olivier@olivier.ms`, Gerhard P. Hancke `Gerhard.Hancke@up.ac.za`

seen, if the observation does not take these layers of description (and self-description) into account.

We have known about this use of metadata in a database for about four decades [3]. The ANSI/SPARC intentional/extensional model of databases divides the database into four layers. Each layer contains some data relevant to that layer; this forms the intentional part of the layer. Each layer (apart from the bottom layer) also contains information that describes the lower layer, forming the extensional aspect of this layer. The four abstract layers are the data model, data dictionary, application schema and application data layers. We have suggested in previous work [2] that it may be useful to use these four layers to conduct a forensic examination. Rather than attempting to determine the impact of the interlinked metadata on the behaviour of the DBMS it was suggested that one could assemble a test configuration of the DBMS in the laboratory and observe behaviour. How and why one would choose a specific version of a layer will be considered in more detail below. In previous work we have also shown that it is indeed possible to split a popular 'real' DBMS into these four layers and reassemble them (with some layers sourced from an earlier version of the database, or recreated for this purpose) into a functioning DBMS [4].

The current paper focuses on forensic examination of the application schema layer. It considers requirements posed by this layer and forensic approaches that are useful to examine this layer. We are not aware of any previous database forensic research that focused on the application schema of the DBMS.

The application schema describes the physical structure of the data [5]. A single database can hold multiple application schema objects such as tables, views, triggers, procedures, indexes, keys and other database designer objects [6]. The application schema is used by more database users on a daily basis than the higher abstract layers (data model and data dictionary). For example, application schema objects like tables are known to more database users than the code of the data model. The larger number of database users on the application schema layer presents more opportunity for the application schema to be altered or damaged. Therefore it is expected that a forensic investigation will be required more often on the application schema layer than higher abstract layers.

Once it is determined that a forensic examination of the application schema is required, the application schema should be prepared to create an environment where the application schema can be investigated. As noted, preparation of the forensic environment is important because the metadata of the DBMS may be altered by an attacker to influence the results of a query [7]. In other words, if the forensic environment is not prepared correctly, the DBMS may deliver incorrect results which may lead to erroneous evidence.

How can we acquire trusted evidence from the DBMS by preparing the application schema for a forensic investigation? This paper will answer this question by firstly describing why a forensic investigation on the application schema level could be required. Several scenarios will illustrate how the application layer could be damaged or altered to show why a forensic investigation could be required on the application schema layer. Thereafter, the clean and found forensic environments will be introduced. Both the found and clean environments will be justified by discussing the advantages of each environment. The following section will define several methods to illustrate how to prepare the application schema for a forensic investigation. Several methods will be discussed in order to give the forensic investigator a variety of options to choose from. Finally, a structured process will be defined for the preparation of a forensic investigation environment on the application schema of the DBMS. The process will take into account the various steps that the investigator needs to follow in order to prepare a forensic environment where the results from the DBMS can be trusted and consequently used for evidence in a court of law. This forensic process will be the result of this study.

2 BACKGROUND

Modern businesses are forced to manage their information in order to survive in the digital era [8]. Organisations without a proper database system are rare. Organisations increasingly rely on information systems (IS), which results in an increase in the use of databases. Databases have become an essential part of all industries today [9]. Computers and other electronic devices are increasingly also used to commit crimes against persons, organizations or even property [10]. Therefore it is a concern that database forensics still lags behind significantly [11].

Database forensics is an emerging field which focuses directly on the identification, preservation, and analysis of the database data suitable for presentation in a court of law [12]. In database forensics it is important to be able to retrace operations performed, reconstruct deleted information, or reconstruct compromised data on the database. In a recent study the field of database forensics was divided into three reconstruction dimensions [11]. The dimensions of database forensics include compromised databases where the metadata or code of the DBMS was tampered with, damaged databases where parts of the DBMS was deleted or damaged, and modified databases where the DBMS changed due to normal operations since the event of forensic interest occurred.

A 'compromised database' is defined as a DBMS where the metadata or software of the DBMS has been modified by an attacker and the DBMS is still functioning. The best tool for collecting data for a forensic analysis may be the DBMS, but the integrity of the DBMS cannot be trusted if the DBMS was compromised to give false information [2]. An incident response study encourages the use of a checklist during the performance of a live response on Oracle databases [13]. This checklist included various tasks to confirm that the Oracle DBMS has not been compromised. A previous practical study by the authors of this paper discussed a method how the metadata could be assembled in order to conduct a forensic examination on a compromised DBMS [4].

A 'damaged database' is defined as a DBMS that has been damaged by deleting, editing or moving the data contained within the DBMS or data files of the DBMS. Most of the research on database forensics falls into this category. Technical methods were proposed to identify when data on an Oracle database has been modified and methods were proposed to recognize vulnerabilities in the Oracle database [14]. Incident responses that relate in some way to database forensics were conducted in corporate environments where clients asked professionals to investigate their databases for possible attacks [15]. A series of papers describe practical methods to recover data from an Oracle database by making use of various sources [16], [17], [18], [13], [19], [20], [21]. This information is very specific towards the Oracle database management system and interlinked with database security. In other literature data mining techniques have been proposed to simplify forensic investigations on large datasets [22]. The preceding information about damaged databases confirms that the research done on database forensics is divergent and often only partially focuses on database forensics.

A 'modified database' is defined as a DBMS that has not been compromised or damaged but has undergone changes due to normal business processes since the event of forensic interest occurred [11]. A modified database will typically be a database which is not directly involved in a crime being investigated, but rather assists in solving the crime. Some work has been published on reconstructing a database to an earlier time even though modifications on the database have occurred [23].

The three dimensions of database reconstruction can be envisioned as being orthogonal. A database being investigated may belong to one or more dimensions in this orthogonal structure. For example, a database being investigated might resemble a damaged database with only some elements of a compromised database and no elements of modified database. In this study our discussion will stretch over all three dimensions of reconstruction in database forensics.

As noted above, a DBMS may be seen as consisting of various layers of metadata and data, each of which serves a different purpose in the system. For example, the executable code in the DBMS serves a completely different purpose than the data stored in the tables of a database. This concept was identified by the ANSI/X3/SPARC Study Group as early as 1975 [3]. The ANSI/X3/SPARC Study Group introduced a model which is now known as the ANSI SPARC database model where DBMSs can be divided into different layers. Based on the ANSI SPARC model, we divided DBMS into four abstract layers. They are the data model, data dictionary, application schema and application data layers.

The first abstract layer of the DBMS is the data model which is a type of metadata and can be viewed as the source code of the DBMS. The second layer of the DBMS is the data dictionary which is the metadata that applies to all databases of the DBMS. The application schema includes user-created operations that can manipulate data such as database triggers, procedures, and sequences, as well as the logical grouping of database objects such as views and tables. The application schema metadata is relevant to one database only as opposed to the data dictionary metadata which is applicable to all databases. The fourth abstract layer is the application data which is the rows stored within the tables of a DBMS.

In a previous study we explored the forensic concepts and methods that apply to the data model layer [7]. This paper extends that work to forensic concepts and methods that apply to the application schema layer. The application schema layer borders with the application data and data dictionary layers. The dividing lines between the layers need to be clear. The application data layer consists of the raw data that is stored within the application schema structures. It may be said that application data rows are stored in application schema tables. The dividing lines between the application schema and data dictionary are more complex. The application schema only consists of data and metadata in the system that is relevant to one database. For example, a trigger is a structure that will always be stored in a single database; therefore a trigger is part of the application schema. The same principle applies to other application schema structures like tables, views, indexes etc., which only apply to a single database. Looking deeper into the DBMS, the data dictionary is made up of metadata which applies to more than one database.

3 REASONS FOR A FORENSIC EXAMINATION OF THE APPLICATION SCHEMA

This section discusses some of the scenarios where a forensic examination of a database's application schema might be useful. Examples are given of how the application schema can be damaged or altered. The alteration or damage to the application schema then warrants a forensic investigation to discover what happened on the DBMS.

It will generally not be possible to tell in advance which of the database layers need to be examined; in fact more than one layer may yield useful evidence. However, in order to explore the topic systematically it seems worthwhile to view each of the layers in isolation, before a multi-layer case is considered. The data model layer has already been explored in this regard [7]; other layers still need to be considered in order to achieve a comprehensive view of DBMS forensics.

There are various reasons why a malicious user may want to transform the database application schema in a way that requires the application schema to be investigated. Customer loyalty programs, targeted marketing, customer services, and the management of practically every corporate resource depend on databases and there are endless reasons why a user may intentionally change, damage, or delete application schema structures [24]. Users may also unintentionally alter or damage the application schema and a forensic investigation will be required to determine what happened.

Alterations or damage to the application schema poses various risks to the owners of the DBMS. These risks include damage to the reputation of a company, financial loss, and hiding of information. The reasons why a person would want to cause harm to the owners of a DBMS include, but are not limited to, vengeance, greed or financial gain. The application schema could be the perfect layer for attacks where the attacker could change data in such a way that the modifications are hard to find unless the investigator knows what to look for. The fact that changes to the application schema may, when properly executed, indirectly affect results of queries makes it inherently harder to trace to the source.

Under normal conditions the application schema is modified to extend and maintain the database. The possible malicious alterations to the application schema are divided into categories below to illustrate the manner in which such modifications may manifest themselves. The scenarios discussed are by no means an exhaustive list of possible alterations to the application schema that might warrant a forensic investigation. There are too many possibilities to mention and a list of such a nature will quickly become outdated and incomplete due to the rapid evolution of databases. We proceed with a discussion of some illustrative scenarios.

3.1 Damaging the application schema

The first category of application schema alterations that might warrant a forensic investigation is direct damage to the application schema. The scenarios mentioned in this section relate to application schema alterations that manifest as damage to the application data layer; the application schema structures need to be recovered in order to access the application data in its structured form. Scenarios include damage to the schema with a SQL drop command, altering access privileges and corrupting tables.

3.1.1 Damage Schema with the SQL Drop Command

A database user might use a SQL command such as *DROP* to remove a table, column, keys or indexes to damage the schema. For example, a table drop of the user table on a web database server will deny users from logging in and cause dissatisfied customers. Even if backups of the database were kept, it might take some time to rebuild the table. Further, the company may not have a way to bill users anymore if backups were not kept of the users' table. Therefore damage to the application schema by commonly used commands is the first reason why an application schema forensic investigation may be required.

3.1.2 Alter Access Privileges

Access privileges are granted and revoked at regular intervals on databases [25]. Every time a new user or web server needs access to an application schema structure, the database authenticates whether the user has access to the application schema structure. The application schema can be altered to disallow a user access to a table structure. The following command can be used to revoke the 'select' rights of a web server on a services table.

```
REVOKE SELECT ON Services FROM
    webserveruser;
```

When a web server loses connectivity to a table which it depends on for crucial data, the owner of the database may experience great losses during the web server down time. A forensic investigation will be required to determine what has gone wrong in the application schema.

3.1.3 Corrupt Table

Another manner in which the application schema could be damaged is to alter the metadata of the DBMS to influence the application schema. A table can be corrupted by changing crucial metadata. The metadata repository can be defined as data which describes the data warehouse itself [26]. The metadata includes the description of the data model, database design definitions, and descriptions of transformations and aggregations. Some of this metadata is directly linked to the application schema of the database. For instance, a database table is defined by making use of metadata. Further, aggregation functions are mapped in metadata and directly influence the way queries are performed on the application schema. This metadata can be altered to change results of queries.

A PostgreSQL DBMS was used to test how a table can be corrupted via metadata. The metadata which constructs a table resides in the pg_class and pg_attribute metadata tables of PostgreSQL. This metadata can be altered to corrupt a table in such a manner that the table displays an error message once a SQL command is run on that table. Various possibilities exist on how to corrupt the table. Values relevant to the table can be deleted from the pg_attribute metadata table, a column can be altered to belong to another table in pg_attribute, the data type of a column can be changed, the corrupted table's columns can be deleted from the pg_class metadata table, etc. If a table is corrupted and cannot be queried anymore it might incur huge losses to the owner of the data. An investigation might be required to determine what has happened, who was responsible, and what the goal of the alteration was (to damage the data, an honest mistake, for personal gain, etc.).

3.2 Application Schema Alterations to Deliver Wrong Results

The next category of alteration to the application schema that might warrant a forensic investigation is alterations that make the DBMS deliver erroneous results. The scenarios include database column swapping, database operator swapping, creating a view to replace a table and damage to the aggregation functions of a database.

3.2.1 Column Swap

The first application schema alteration that could influence the DBMS to deliver erroneous results is to swap a column in the metadata [2]. The authors of this paper compromised the application schema by swapping two column names within a table [4]. In PostgreSQL the *attnum* sequence in the *pg attribute* table was changed for these two columns. This means that if a select query is executed on the compromised table using the name of one column, the values of the other column will be returned. Figure 1 illustrates what commands can be used to swap two columns in the PostgreSQL DBMS. The *pg_attribute* table consists of metadata which constructs the tables of the PostgreSQL DBMS. If this metadata is changed, the application schema is altered significantly. A forensic investigation might be required to examine an application schema alteration of this nature.

```
update pg_attribute set attnum = '4' where attrelid =
    '16388' and attname = 'number';
update pg_attribute set attnum = '2' where attrelid =
    '16388' and attname = 'highnumber';
update pg_attribute set attnum = '3' where attrelid =
    '16388' and attname = 'number';
```

Figure 1: Commands used to swap two columns of a table in the application schema.

3.2.2 Operator Swap

Another modification of the application schema metadata that may warrant a forensic investigation is an operator swap. This causes the DBMS to yield incorrect results on a query that uses the affected operator. The DBMS makes use of metadata to define what processes should be used to perform mathematic operator calculations. The metadata, for example, links the plus operator (+) to an addition processes. Such metadata can be changed to link an operator to the wrong process. For example, the division operator might be changed to make use of the multiplication process. Every time the division operator is used, a multiplication will be carried out. This is defined as an operator swap. Figure 2 illustrates a command that may be used to alter a division operator to execute multiplication in PostgreSQL.

```
update pg_operator set oprcode = ‘int2mul’ where
    oprname = ‘/’ and oprleft = ‘21’ and oprresult =
    ‘21’;
```

Figure 2: Commands used to alter the division operator to multiply.

3.2.3 Create View to Replace Table

Creating a view to mimic a table to hide data is another way the application schema can be modified to yield incorrect results. Imagine a situation where a table with the name *MinimumCost* exists. The table is mostly used to perform *SELECT* queries and is rarely updated. The table is then renamed by some party to *MinimumCost2* and a view is created with the name *MinimumCost* to replace the table. However the table can be modified to hide some rows from the original table. The following SQL command can, for example, be used to create the view to hide some rows from the initial table.

```
CREATE VIEW MinimumCost AS SELECT *
    FROM MinimumCost2 WHERE ID <> 858;
```

3.2.4 Aggregation Damage

DBMSs use aggregation functions in SQL commands to calculate and summarize data [27]. These aggregation functions are captured in the metadata in a similar way to the mathematic operators mentioned above. The pg_aggregate table in PostgreSQL holds the metadata about how to handle aggregation functions [28]. The values of this table can be altered to cause damage to an aggregation function. The pg_attribute table matches an aggregation function with a function that handles the aggregation function calculation. An aggregation function can then be mismatched with an incorrect function. For example, the SUM aggregation function can be altered to make use of the AVG function calculation processes and therefore deliver unreliable results. Aggregation functions are frequently used in databases and all the queries that make use of the damaged aggregation function will return incorrect results. Figure 3 displays how an aggregation function can be altered in the PostgreSQL database. After this alteration the SUM aggregation function will return incorrect and unpredictable results, but no error is given which will require someone to work out the actual sum to figure out that the database is sending incorrect results.

3.3 Database Behaviour Alterations

This section illustrates why a forensic investigation may be required due to alterations to the application schema that change the behaviour of the database. The examples discussed below include slowing the DBMS down by dropping table indexes and modifying the storage engines of a table.

```
update pg_aggregate set aggtransfn =
    'int4_avg_accum'::regproc where aggtransfn =
    'int4_sum'::regproc;
update pg_aggregate set aggfinalfn =
    'int8_avg'::regproc, aggtranstype = 1016,
    agginitval = '{0,0}' where aggtransfn =
    'int4_avg_accum'::regproc;
```

Figure 3: Commands used to alter an existing aggregation function.

3.3.1 Slow the DBMS Down by Dropping Indexes

The first way to change the application schema in order to affect the behaviour of the DBMS is to drop the indexes from a table and slow down large database searches tremendously. A dropped index is something that may not be easily be detected by the system administrator and may cause damage to the reputation of a customer-facing database software product. Even if the database administrator eventually finds the cause of the problem, a forensic investigation may still be required to determine what happened.

3.3.2 Blackhole Storage Engine

Another method of altering the application schema which might warrant a forensic investigation is the use of different storage engines. MySQL supports various storage engines which act as handlers of different table types [29]. By default the MyISAM engine is used in MYSQL, but the storage engine can be changed by altering a table in the application schema. The storage engine can be altered and consequently change the behaviour of the database. The blackhole storage engine is used in scenarios where there are a large number of database slaves plus a busy master machine [30]. Network load can become significant even when compression methods are used. The blackhole storage engine acts as a blackhole that accepts data, but discards rather than stores it. For example, if a user inserts 100 values into a table, the database will accept the insert command and report a successful insertion, but no values will in fact be inserted into the table. This engine might enable a malicious user to prevent a database from updating information. This example also demonstrates that some features in particular DBMSs may be used to exploit the application schema.

3.3.3 Custom Storage Engines

Yet another threat stems from the fact that database experts can write their own storage engines to perform potentially malicious tasks. Such a custom storage engine can then be activated in the application schema of the database. The coding of the custom storage engine takes place in the data model abstract layer when the code of the DBMS is changed, but the application schema is changed to activate the custom storage engine. More information on writing a custom storage engine can be found in [31].

4 SELECTING AN APPLICATION SCHEMA FORENSICS ENVIRONMENT

The previous section discussed some practical examples of when application schema forensics might be required. This section focuses on how to approach a forensic examination of an application schema. Environments are introduced where a forensic investigation can be conducted in and several advantages of the various environments will be mentioned. A forensic environment should be selected that will best fit the forensic context.

For any given layer, two obvious alternatives exist when the DBMS is assembled for forensic testing. One alternative is to use a layer as it was *found* on a DBMS server to be examined. The other alternative is to use a *clean* version of the layer—that is, a version that is 'correct'. For some layers this is usually rather simple. It has been noted that the data model layer in practice consists of the DBMS software. To obtain a 'clean' version of this software to examine the underlying files is not problematic (although care may have to be taken to ensure that the clean version is patched to the same extent as the software it replaces has been patched). In fact, found and clean options are often not the only options: An examiner may, for example, expressly decide to use an unpatched version of the software if the specific examination requires it.

A found version of the application schema layer can readily be obtained. One option is to image the server's hard drive (or equivalent) and obtain the relevant data from the image. However, it may often not be possible to find an 'original' (or *clean*) version of such a schema. Exceptions exist when the design documentation of the database is available and no changes have been made to the original database, or authorised changes have been meticulously documented through a proper change management process. If a clean version is not available, a backup that was made before any suspected malicious activity occurred may have to be used as a 'clean' version. In fact, as noted above, *found* and *clean* are not the only options—a forensic examination may be focussed on changes that may have been made between two dates and layers extracted from backups made on (or near) those dates may be appropriate choices to use in the reconstructed DBMS. The expertise of the examiner and the purpose of the examination will determine choices made. The examiner should be able to justify such choices and note the effect of choices on the outcome of the examination. For the sake of simplicity we proceed with *found* and *clean* as the only choices, noting that these alternatives will in many cases be the important ones. However, when these two are used below the reader should recall that other alternatives exist.

The following sections will discuss why it might be necessary to have a found and/or clean layer for a forensic investigation.

4.1 Indications for using a clean layer

Using a clean layer ensures that the layer will not distort data (or metadata) obtained via it; it will present any results obtained via it as originally intended by the designers. Phrased differently, a clean layer is chosen or constructed to behave exactly like it did in the DBMS being examined, but we are certain that it is free from modifications or other artefacts that may corrupt output retrieved from the DBMS. A clean data schema layer has several uses when a forensic examination of a database is conducted. Some of these uses are discussed in the following sections.

4.1.1 Application Schema Structure Recovery

The first reason for using a forensic environment where parts of the application schema are clean (or have been cleaned) is the possibility to repair application schema structures. If an application schema structure (like a table) is damaged on the DBMS being examined, it may be helpful to recreate that application schema structure on a replica DBMS in order to query the table 'naturally'. Note that it is risky to attempt to repair application schema structures on the live environment since evidence cannot be altered. Using a replica of the original minimises the risk of damaging evidence when repairing application schema structures.

4.1.2 Clean Metadata

The second reason for using a clean application schema layer is that a comparison between results derived from a replica with a such a clean layer and the original may reveal differences that have forensic value. More specifically, such differences may help to isolate and determine the extent of modifications made to this layer as it was being used. Clean metadata may also reveal application schema structures that have been hidden in some manner during use of the database. Additionally, clean metadata may restore basic DBMS functions that enable the examiner to use such functions when searching for forensic evidence. If the layer is only partially cleaned, it should be clear what metadata (cleaned or found) is used whenever a forensic query is executed. Note that cleaning metadata tables such as pg_attribute in PostgreSQL will cause all table information to be lost and these tables will have to be recreated.

4.1.3 Integrity

A DBMS can be modified in various ways to produce results that are unusual or unexpected. As mentioned previously in this study, the metadata of a database can be changed to deliver erroneous results or an application schema structure can be corrupted not to deliver results. A clean environment provides a degree of integrity where the investigator can be sure that evidence results are not influenced by modifications made to the application schema. Metaphorically, a clean data application layer provides a 'clean lens' through which the application data may be inspected.

4.1.4 Elimination of Layers

Imagine a situation where the database is producing unpredictable (or consistently incorrect) results due to a modification made to a particular layer of the DBMS. An investigator may take an educated guess about which layer is causing the unpredictable results. However, in many cases the various layers of the DBMS will need to be eliminated as the cause of the problem. This elimination can be done by using clean versions of the various layers (where available) one at a time, with the other layers as found to determine if the unpredictable results persist. A clean layer may therefore assist to find the source of a problem through a process of elimination.

4.2 Arguments for a found environment

A found environment is one that reflects the state of the DBMS at the time the investigation started (or, if possible, when the event of forensic interest occurred). A found environment exists, by definition, on the live server (but may have to be preserved) if examination cannot start immediately. The term *found* is also used to refer to a copy of a layer that was obtained from the DBMS at such a time, and where the copy is used for examination as part of a replica in the laboratory. It is important to understand that the found environment is not precisely the same here as the traditional meaning of a live digital forensic environment, due to the fact that the environment may fully or partially exist on the live machine or on another machine. Several reasons for using a found application schema layer for forensic examination are discussed in the following sections.

4.2.1 Cost of Forensic Investigation

The first argument to use a found forensic environment is the fact that it may decrease the cost of the investigation. There are a couple of reasons why a forensic investigation on a found environment is may be cost-effective:

1. expertise is not required to prepare the clean forensic environment,
2. institutions can barely afford the time required to prepare a new forensic environment.

When there is little reason to think that a DBMS layer could have been modified, using the found environment could yield results much quicker.

4.2.2 Test Hypothesis

A hypothesis can be tested in the found environment without making any changes to the environment of the DBMS. Sometimes in digital forensics a hypothesis needs to be tested based on a hunch of the forensic investigator. The found environment provides an easy accessible platform to test hypotheses. Here we are referring to leads that may be useful for further investigation, rather than extraction of evidence for court use.

5 METHODS TO DEAL WITH APPLICATION SCHEMA FORENSICS

The previous section argued why a clean or found environment may be useful in a forensic investigation. In this section we will discuss how application schema forensic methods can be applied. This section is divided into a discussion of methods to achieve a clean environment and methods to achieve a found environment for a forensic examination of an application schema. Several methods will be suggested for the clean and found environments. Finally we briefly discuss some known evidence-searching techniques.

5.1 Clean Application Schema Environment

A clean application schema environment is achieved by establishing a new forensic environment on a different database server in order to determine what has been done on the database or to determine why the database is behaving in a particular way. The following sections discuss methods to achieve a clean application schema environment.

5.1.1 Rebuild from Previous Dumps

One method to achieve a clean application schema environment is to rebuild the application schema from previous dumps of the database of interest. The dump file to be used should have been created before the event of forensic interest occurred. Dump files include an application data section which is generally represented as 'insert' SQL commands, and an application schema section which is generally identified by the 'create' SQL commands. The SQL commands in the dump file which affect the application schema can be used to rebuild the application schema on a new database server. Either a whole clean installation should be made or the current database installation should be copied over from the live database server depending on the state of the data model layer. Since our focus is on the application schema it will be discussed in isolation in the following sections. Lastly, an old dump of the application schema should be applied to the database which will be investigated. Thereafter the application data of the live database should be dumped and that data should be applied to the clean application schema. This method does not take a lot of time, expertise or effort.

5.1.2 Rebuild According to Documentation

If previous dumps are not available, database design documentation may be used to rebuild the application schema. This is the second method to achieve a clean application schema forensic environment. Database design documentation may be available in a variety of forms. These forms include an entity-relationship (ER) model, database design software that was not synchronised with the application schema of forensic interest (such as MySQL Workbench) or official sketches or diagrams [32]. The application can be rebuilt by running SQL commands to restore the application schema according to the available documentation. After the application schema has been rebuilt, a data dump of the live database can be applied to the clean application schema. The amount of effort required to apply this method depends on the size of the application schema and the type of documentation available. Large application schemas will require a lot of effort to be rebuilt from an ER Diagram, but will not require much time when using database design software, which can build the application schema automatically.

5.1.3 Rebuild from Pieces

Another method to construct a clean application schema forensic layer is to rebuild the application schema from pieces after an event has caused major damage to the database. This method is particularly relevant to cases where the DBMS has been destroyed to some extent and a forensic investigation is warranted. During such an event data may be lost and database structures destroyed. The application schema might be rebuilt from the leftover data to varying degrees depending on what could be recovered. This leftover data will typically be found on the file system of the server where the DBMS was installed. In this scenario the console of the DBMS might not even work anymore and the application schema should be retrieved on the file system because no data dumps or application schema reports are available from the DBMS console. The relevant DBMS installation files should be understood and the required DBMS files from the file system should be copied into a working version of the DBMS. In this way some of the application schema data may be retained and a better understanding can be attained on the state of the application schema before the DBMS was destroyed. This method requires a significant degree of expertise and effort but might be the only forensic option in some cases.

5.1.4 New Metadata Template and Metadata Alterations

The final method to obtain a clean application schema layer is to replace the metadata of the application schema with clean metadata. Some DBMSs (like PostgreSQL) make use of metadata to store data about application schema structures. This data can be changed in order to change the structure and behaviour of the DBMS. If it could be determined that the metadata of the application schema is the cause of suspect DBMS behaviour, then parts of the metadata template may be replaced with clean chunks of metadata in order to test the hypothesis. With this method a copy of the application schema should be set up on another database server (not on the live database server) and the metadata of that DBMS should be changed according to the requirements of the forensic investigation. The application schema can now be tested to either reveal forensic evidence or to confirm a hypothesis. This is an advanced method that requires a lot of expertise on the particular DBMS.

5.2 Found Application Schema Environment

The found application schema environment is a setting where the application schema could be forensically examined as it was found by the forensic investigator. As opposed to the clean environment, the found application schema environment does not make use of methods to clean the application schema in any way. The evidence should stay the same on the application schema level. The found environment is similar to a post-mortem forensic environment, but a found environment may be mirrored onto another machine and the forensic investigation could commence on that machine. The following sections will discuss methods to achieve a found application schema forensic environment.

5.2.1 Live Environment

The first method to achieve a found application schema environment is to leave the application schema untouched on the database where the incident of forensic interest occurred. The application schema will still be in a live state and a typical live investigation needs to be done on the application schema. During a live investigation the best practices should be used to leave the evidence unchanged as far as possible. This scenario may also be useful in circumstances where a low cost forensic investigation is a high priority.

5.2.2 Copy of the Application Schema

Another way a found environment can be set up for a forensic investigation is by mirroring the application schema onto another installation of the database. Note that in this scenario the entire application schema is mirrored after the incident of forensic interest occurred. This method could be achieved by dumping the live application schema and importing the application schema into the desired database environment. This method becomes complicated when we suspect that the data model has been affected by the incident and we can no longer trust the data model's dump function. In this instance the application schema will need to be copied, along with the application data, from the data files on the operating system. This method is particularly useful when the behaviour of the application needs to be tested or simulated.

5.3 Techniques to find evidence in the forensic environment

At this stage of this study we have discussed the advantages of the found and clean environment; and we discussed how a clean or found application schema environment can be achieved. In the following sections we will mention some approaches to identify evidence during an investigation once either of these environments has been established.

5.3.1 Find Inconsistencies

One method to identify application schema evidence is to find inconsistencies in audit trails. Various sources of data can be used as the audit trail and they include (and are not limited to) the database table, log files and application data.

5.3.2 Output Based Investigation

Another approach to an application schema forensic investigation is to search for evidence based on the output of the DBMS. For example, if a certain application schema structure returns unpredictable values, the particular application schema structure and closely related structures can be targeted for forensic evidence.

5.3.3 Find the origin of the problem

This approach overlaps with the two approaches to an application schema forensic investigation mentioned above, but is worth mentioning here. In a forensic investigation it is critical to find the source of the problem in order to investigate the surroundings for evidence. For instance, if a table returns wrong results on an aggregation query the logical path to take will be to check the query, check the data of the table, check the table structure and check triggers or procedures running on the function. If no evidence is found at this stage, then the aggregation function of the DBMS should be investigated to ensure it is still working. If the origin of the problem was found at the aggregation function itself, then evidence directly related to the aggregation function and aggregation tables can be targeted.

6 APPLICATION SCHEMA FORENSIC PREPARATION PROCESS

In this section an application schema forensic preparation process will be introduced. As illustrated by Cohen [33], the entire forensic process to obtain evidence from digital devices, from the identification of evidence to presentation in court, is an elaborate one. Here the focus will be on one of the steps that the authors of this paper consider one of the most important parts of the forensic process: the forensic preparation process. Figure 4 illustrates this process as a flow diagram.

The process can be divided into three key areas:

1. choose forensic environment,
2. select and implement method to achieve the environment,
3. extract evidence.

These key areas are ordered in a logical way and need to be executed consecutively as illustrated in Figure 4.

The selection of a forensic environment is made by first considering the crime scene or forensic setting where the evidence should be extracted from. These settings may vary significantly. As mentioned earlier, the selection of a suitable environment is a vital part in ensuring the integrity of the evidence we receive from the DBMS. A clean environment may be impractical or unnecessary in some settings and the only option in other settings. Equally, the found environment may be

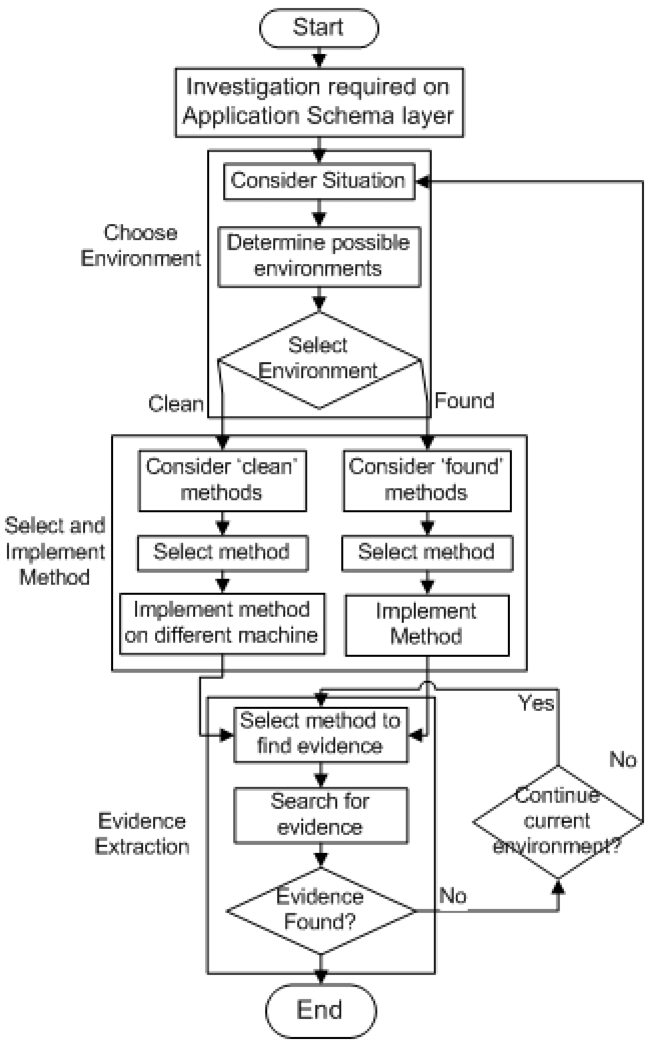

Figure 4: The Application Schema Evidence Identification Process.

an incorrect choice in some settings and the only feasible choice in other settings. The forensic investigator needs to choose one of these two environments.

Once a forensic environment is selected we need to select a method on how to transform the forensic setting into the selected forensic environment. It is important to select a method to transform the crime scene into the correct forensic environment because our actions should not affect the evidence in the DBMS. Methods should be used which sustain the integrity of the evidence from the DBMS. Several methods have been discussed in this paper on how to achieve a found or clean application schema environment. The type of method will be dependent on the environment that was selected for the investigation. Only found methods can achieve a found forensic environment and only clean methods can achieve a clean forensic environment. The method most suited for the forensic investigation should be selected. Thereafter the method should be implemented.

Once the forensic environment has been prepared we are ready to start searching and identifying forensic evidence in this environment. Typical forensic evidence searching methods may be used at this stage. Some of them were discussed earlier in this paper. If forensic evidence cannot be found in the current forensic environment the forensic environment can be changed and the process started over again. Alternatively, if evidence was found, the application schema evidence identification process is completed and the investigator can move on to collecting the evidence, preserving the evidence, transporting the evidence, etc.

7 CONCLUSION

The application schema layer of a DBMS can be altered in various ways to achieve a wide variety of results that may warrant a forensic investigation. The found and clean forensic environments are two forensic environments that can assist a forensic investigator to identify evidence in the application schema layer of the DBMS. The investigator should understand the advantages of each environment and be familiar with the methods required to implement these environments. This study concluded with a process that will apply to various different forensic scenarios of the application schema. The application schema forensic evidence identification process will assist an investigator in making good decisions while conducting a forensic investigation on the application schema layer of the DBMS. Future work will bring all layers of the DBMS together in order to design an evidence collection process that includes all layers of the DBMS.

REFERENCES

[1] E. Chickowski. "Database forensics still in the dark ages", 2011. URL `http://www.darkreading.com/attacks-breaches/database-forensics-still-in-dark-ages/d/d-id/1136132`. Last accessed 15 Nov 2014.

[2] M. S. Olivier. "On metadata context in database forensics". *Digital Investigation*, vol. 5, no. 3–4, pp. 115–123, 2009.

[3] ANSI/X3/SPARC Study Group. "Data base management systems: Interim report". *ACM SIGMOD bulletin*, vol. 7, no. 2, 1975.

[4] H. Beyers, M. S. Olivier and G. P. Hancke. "An approach to examine the metadata and data of a database management system by making use of a forensic comparison tool". In H. S. Venter, M. Coetzee and M. Loock (editors), *Proceedings of the 2011 Information Security for South Africa (ISSA 2011) Conference*. Johannesburg, South Africa, 8 2011. (Work in Progress Paper; published electronically).

[5] R. M. Riordan. *Designing effective database systems*. Addison-Wesley Professional, Massachusetts, USA, 2005.

[6] C. Coronel, S. Morris and P. Rob. *Database Systems: Design, Implementation, and Management*. Cengage Learning, Boston, USA, 2009.

[7] H. Q. Beyers, M. S. Olivier and G. P. Hancke. "Arguments and methods for database data model forensics". In *Seventh International Workshop on Digital Forensics & Incident Analysis (WDFIA)*, pp. 139–149. Hersonissos, Crete, Greece, 6 2012.

[8] E. Fernandez-Medina and M. Piattini. "Designing secure databases". *Information and Software Technology*, vol. 47, pp. 463–477, Nov. 2005.

[9] S. Sumathi and S. Essakkirajan. *Fundamentals of Relational Database Management Systems — Studies in computational intelligence.* Springer, 2007.

[10] J. Ashcroft. "Electronic crime scene investigation: A guide for first responders". NIJ Guide NCJ 187736, National Institute of Justice, Office of Justice Programs, U.S. Department of Justice, 2001.

[11] O. M. Fasan and M. S. Olivier. "On dimensions of reconstruction in database forensics". In *Seventh International Annual Workshop on Digital Forensics & Incident Analysis (WDFIA)*, pp. 97–106. Hersonissos, Crete, Greece, Jun. 2012.

[12] K. Fowler. *SQL Server forensic analysis.* Addison-Wesley Professional, NJ, USA, 2008.

[13] D. Litchfield. "Oracle forensics part 4: Live response". Insight security research publication, NGSSoftware, Apr. 2007.

[14] P. M. Wright. "Oracle database forensics using Logminer". Global information assurance certification paper, SANS Institute, Jan. 2005.

[15] K. Fowler. "Forensic analysis of a SQL Server 2005 database server", Apr. 2007. URL `http://www.sans.org/reading_room/whitepapers/application/forensic-analysis-sql-server-2005-database-server_1906`. Last accessed on 15 Nov 2014.

[16] D. Litchfield. "Oracle forensics part 1: dissecting the redo logs". Insight security research publication, NGSSoftware, Mar. 2007.

[17] D. Litchfield. "Oracle forensics part 2: Locating dropped objects". Insight security research publication, NGSSoftware, Mar. 2007.

[18] D. Litchfield. "Oracle forensics part 3: Isolating evidence of attacks against the authentication mechanism". Insight security research publication, NGSSoftware, Mar. 2007.

[19] D. Litchfield. "Oracle forensics part 5: Finding evidence in the absence of auditing". Insight security research publication, NGSSoftware, Aug. 2007.

[20] D. Litchfield. "Oracle forensics part 6: Examining undo segments, flashback and the Oracle recycle bin". Insight security research publication, NGSSoftware, Aug. 2007.

[21] D. Litchfield. "Oracle forensics part 7: using the Oracle system change number in forensic investigations". Insight security research publication, NGSSoftware, Nov. 2008.

[22] N. Beebe and J. Clark. "Dealing with terabyte data sets in digital investigations". In *Proceedings of the IFIP International Conference on Digital Forensics.* Orlando, USA, February 2005.

[23] O. M. Fasan and M. S. Olivier. "Reconstruction in database forensics". In G. Peterson and S. Shenoi (editors), *Advances in Digital Forensics VIII*, pp. 273–287. Springer, 2012.

[24] E. Oz. *Management information systems.* Course Technology, Boston, USA, 6th edn., 2009.

[25] B. Thomas. *OCA: Oracle database 11g Administrator Certified Associate study guide.* John Wiley and Sons, Indianapolis, USA, 2010.

[26] P. Ward and G. Dafoulas. *Database management systems.* Thomson Learning, London, UK, 2006.

[27] R. Narang. *Database management systems.* PHI Learning, New Delhi, India, 2006.

[28] D. L. et al. *The database hacker's handbook.* Wiley, Indianapolis, USA, 1st edn., 2005.

[29] MySQL AB. *MySQL Administrator's Guide and Language Reference.* MySQL Press, Uppsala, Sweden, 2nd edn., 2006.

[30] D. Schneller and U. Schwedt. *MySQL admin cookbook.* Packt Publishing, Brimingham, UK, 2010.

[31] MySQL Forge. "MySQL internals custom engine", 2010. URL `http://forge.mysql.com/wiki/MySQL_Internals_Custom_Engine`. Last accessed on 2 May 2012.

[32] Sideris Courseware Corp. *Data modelling: Logical database design.* Sideris Courseware Corp, Newton, USA, 2011.

[33] F. Cohen. "Fundamentals of digital forensic evidence". In P. Stavroulakis and M. Stamp (editors), *Handbook of Information and Communication Security*, pp. 789–808. Springer, San Jose, USA, 2010.