

**DATABASE FORENSICS: INVESTIGATING COMPROMISED DATABASE
MANAGEMENT SYSTEMS**

by

Hector Quintus Beyers

Submitted in fulfilment of the requirements for the degree
Master of Engineering (Computer Engineering)

in the

Department of Electrical, Electronic and Computer Engineering
Faculty of Engineering, Built Environment and Information Technology

UNIVERSITY OF PRETORIA

May 2013

SUMMARY

DATABASE FORENSICS: INVESTIGATING COMPROMISED DATABASE MANAGEMENT SYSTEMS

by

Hector Quintus Beyers

Supervisor: Prof. G.P. Hancke
Co-supervisor: Prof. M.S. Olivier
Department: Electrical, Electronic and Computer Engineering
University: University of Pretoria
Degree: Master of Engineering (Computer Engineering)
Keywords: Database, Database Forensics, Compromised Database, Database Management System, Data Model Forensics, Data Dictionary Forensics, Application Schema Forensics

INTRODUCTION

The use of databases has become an integral part of modern human life. Often the data contained within databases has substantial value to enterprises and individuals. As databases become a greater part of people's daily lives, it becomes increasingly interlinked with human behaviour. Negative aspects of this behaviour might include criminal activity, negligence and malicious intent. In these scenarios a forensic investigation is required to collect evidence to determine what happened on a crime scene and who is responsible for the crime. A large amount of the research that is available focuses on digital forensics, database security and databases in general but little research exists on database forensics as such. It is difficult for a forensic investigator to conduct an investigation on a DBMS due to limited information on the subject and an absence of a standard approach to follow during a forensic investigation. Investigators therefore have to reference disparate sources of information on the topic of database forensics in order to compile a self-invented approach to investigating a database. A subsequent effect of this lack of research is that compromised DBMSs (DBMSs that have been attacked and so behave abnormally) are not considered or understood in the database forensics field. The concept of compromised

DBMSs was illustrated in an article by Olivier who suggested that the ANSI/SPARC model can be used to assist in a forensic investigation on a compromised DBMS. Based on the ANSI/SPARC model, the DBMS was divided into four layers known as the data model, data dictionary, application schema and application data. The extensional nature of the first three layers can influence the application data layer and ultimately manipulate the results produced on the application data layer. Thus, it becomes problematic to conduct a forensic investigation on a DBMS if the integrity of the extensional layers is in question and hence the results on the application data layer cannot be trusted. In order to recover the integrity of a layer of the DBMS a clean layer (newly installed layer) could be used but clean layers are not easy or always possible to configure on a DBMS depending on the forensic scenario. Therefore a combination of clean and existing layers can be used to do a forensic investigation on a DBMS.

PROBLEM STATEMENT

The problem to be addressed is how to construct the appropriate combination of clean and existing layers for a forensic investigation on a compromised DBMS, and ensure the integrity of the forensic results.

APPROACH

The study divides the relational DBMS into four abstract layers, illustrates how the layers can be prepared to be either in a found or clean forensic state, and experimentally combines the prepared layers of the DBMS according to the forensic scenario. The study commences with background on the subjects of databases, digital forensics and database forensics respectively to give the reader an overview of the literature that already exists in these relevant fields. The study then discusses the four abstract layers of the DBMS and explains how the layers could influence one another. The clean and found environments are introduced due to the fact that the DBMS is different to technologies where digital forensics has already been researched. The study then discusses each of the extensional abstract layers individually, and how and why an abstract layer can be converted to a clean or found state. A discussion of each extensional layer is required to understand how unique each layer of the DBMS is and how these layers could be combined in a way that enables a forensic investigator to conduct a forensic investigation on a compromised DBMS. It is illustrated that each layer is unique and could be corrupted in various ways. Therefore, each layer must be studied individually in a forensic context before all four layers are considered collectively. A forensic study is conducted on each abstract layer of the DBMS

that has the potential to influence other layers to deliver incorrect results. Ultimately, the DBMS will be used as a forensic tool to extract evidence from its own encrypted data and data structures. Therefore, the last chapter shall illustrate how a forensic investigator can prepare a trustworthy forensic environment where a forensic investigation could be conducted on an entire PostgreSQL DBMS by constructing a combination of the appropriate forensic states of the abstract layers.

RESULTS

The result of this study yields an empirically demonstrated approach on how to deal with a compromised DBMS during a forensic investigation by making use of a combination of various states of abstract layers in the DBMS. Approaches are suggested on how to deal with a forensic query on the data model, data dictionary and application schema layer of the DBMS. A forensic process is suggested on how to prepare the DBMS to extract evidence from the DBMS. Another function of this study is that it advises forensic investigators to consider alternative possibilities on how the DBMS could be attacked. These alternatives might not have been considered during investigations on DBMSs to date. Our methods have been tested at hand of a practical example and have delivered promising results.

OPSOMMING

DATABASISFORENSIKA: ONDERSOEK VAN 'N ONBETROUBARE DATABASISBEHEERSTELSEL

deur

Hector Quintus Beyers

Studieleier: Prof. G.P. Hancke
Mede-studieleier: Prof. M.S. Olivier
Departement: Elektriese, Elektroniese en Rekenaaringenieurswese
Universiteit: Universiteit van Pretoria
Graad: Magister in Ingenieurswese (Rekenaaringenieurswese)
Sleutelwoorde: Databasis, Databasisforensika, Onbetroubare databasis,
Databasisbeheerstelsel, Datamodelforensika,
Datawoordeboekforensika, Toepassingskemaforensika

INLEIDING

Die gebruik van databasisse vorm vandag deel van die moderne mens se lewe. Individue en maatskappye heg dikwels baie waarde aan data wat in databasisse gestoor word. Soos databasisse 'n groter deel van mense se lewens uitmaak, word dit al meer aan menslike gedrag gekoppel. Negatiewe aspekte van hierdie gedrag behels dikwels kriminele aktiwiteite, verwaarlosing en skadelike gedrag wat toegepas kan word op databasisse. Tydens sulke situasies word 'n forensiese ondersoek benodig om bewyse op te spoor ten einde te kan bepaal wat op die misdaadtoneel plaasgevind het en wie vir die misdaad verantwoordelik is. Daar is heelwat navorsing beskikbaar oor digitale forensika, databasissekuriteit en databasisse in die algemeen, maar min navorsing is beskikbaar wat op databasisforensika. Dit is moeilik vir 'n forensiese ontleder om 'n ondersoek op 'n databasisbeheerstelsel uit te voer weens die tekort aan inligting en die afwesigheid van standaardbenaderings tydens 'n forensiese ondersoek. Daarom moet forensiese ontleders tans self 'n groot verskeidenheid inligtingsbronne oor databasisforensika bymekaarmaak om self 'n benadering te kies vir 'n ondersoek op 'n databasisbeheerstelsel. 'n Gevolg van hierdie tekort aan navorsing is dat onbetroubare databasisbeheerstelsels (wat aangeval is en

abnormaal reageer) nie oorweeg of verstaan word in databasisforensika nie. Die konsep van onbetroubare beheerstelsels is in 'n artikel deur Olivier voorgestel waar hy verduidelik dat die ANSI/SPARC-model gebruik kan word om te help tydens 'n forensiese ondersoek op 'n onbetroubare databasisbeheerstelsel. Die databasisbeheerstelsel word volgens die ANSI/SPARC-model in vier vlakke verdeel. Dit is die datamodel-, datawoordeboek-, toepassingskema- en toepassingsdatavlak. Die invloedryke aard van die eerste drie vlakke beïnvloed die toepassingsdatavlak en uiteindelik word die resultate vanuit die toepassingsdatavlak gemanipuleer. Dus word dit problematies om 'n forensiese ondersoek op 'n databasisbeheerstelsel uit te voer as die integriteit van die invloedryke vlakke bevraagteken kan word en die resultate van die toepassingsdatavlak nie meer vertrou kan word nie. Om die integriteit van 'n vlak van die databasisbeheerstelsel te herstel, kan 'n skoon vlak (nuutgeïnstalleerde vlak) gebruik word, maar om skoon vlakke op 'n databasisbeheerstelsel te konfigureer, is nie altyd maklik of moontlik nie, afhangende van die forensiese omgewing. Daarom moet 'n kombinasie van skoon en bestaande vlakke gebruik word om 'n ondersoek op 'n databasisbeheerstelsel uit te voer.

PROBLEEMSTELLING

Die probleem wat aangespreek moet word, is hoe om die regte kombinasie skoon en bestaande vlakke bymekaar te voeg vir 'n forensiese ondersoek op 'n onbetroubare databasisbeheerstelsel sodat die integriteit van die forensiese resultate behoue kan bly.

BENADERING

Hierdie studie verdeel die PostgreSQL-databasisbeheerstelsel in vier abstrakte vlakke, illustreer hoe die vlakke voorberei kan word om in 'n skoon of gevonde forensiese toestand te bestaan en kombineer die voorbereide vlakke van die databasisbeheerstelsel volgens die forensiese omgewing. Die studie begin met agtergrondinligting oor die onderwerpe van databasisse, digitale forensika en databasisforensika onderskeidelik om die leser in te lig oor die bestaande navorsing in die relevante velde. Daarna bespreek die studie die vier abstrakte vlakke van die databasisbeheerstelsel en verduidelik hoe die vlakke mekaar kan beïnvloed. Die skoon en gevonde omgewings word voorgestel op grond van die feit dat databasisbeheerstelsels anders is as tegnologieë waar digitale forensika reeds nagevors is. Die studie bespreek dan elke invloedryke vlak individueel asook hoe en hoekom die abstrakte vlak in 'n skoon of gevonde vlak omskep kan word. 'n Bespreking van elke

invloedryke vlak is nodig om te verstaan hoe uniek elke vlak van die databasisbeheerstelsel is en hoe hierdie vlakke gekombineer kan word wat 'n forensiese ondersoekbeampte in staat sal stel om 'n forensiese ondersoek op 'n onbetroubare databasisbeheerstelsel uit te voer. Dit bewys dat elke vlak uniek is en op verskeie maniere onbetroubaar gemaak kan word. Daarom word elke vlak individueel in 'n forensiese konteks bestudeer voor die databasis as 'n geheel bestudeer word. 'n Forensiese studie is op elke abstrakte vlak van die databasisbeheerstelsel uitgevoer wat die potensiaal het om ander vlakke te beïnvloed om foutiewe resultate te lewer. Uiteindelik word die databasisbeheerstelsel as 'n forensiese instrument gebruik om bewyse vanuit sy eie datastrukture na vore te bring. Daarom toon die laaste hoofstuk aan hoe 'n forensiese ondersoekbeampte 'n betroubare forensiese omgewing kan voorberei waar 'n forensiese ondersoek op die hele databasisbeheerstelsel uitgevoer kan word deur die regte forensiese toestande van die abstrakte vlakke te kombineer.

RESULTATE

Die uitkoms van hierdie studie is 'n impiriese benadering wat aantoon hoe om 'n ondersoek op 'n onbetroubare databasisbeheerstelsel uit te voer deur van 'n kombinasie van verskeie toestande van die abstrakte vlakke van die databasisbeheerstelsel gebruik te maak. Benaderings word voorgestel oor hoe om 'n forensiese navraag op die datamodel, datawoordeboek en toepassingskema van die databasisbeheerstelsel te hanteer. 'n Forensiese proses word voorgestel oor hoe om die databasisbeheerstelsel voor te berei om bewyse te verkry. 'n Bykomende funksie van die studie is om forensiese ondersoekbeamptes te adviseer om alternatiewe moontlikhede te oorweeg oor hoe die databasisbeheerstelsel aangeval kan word. Hierdie alternatiewe is moontlik nog nie voorheen tydens 'n ondersoek op 'n databasisbeheerstelsel oorweeg nie. Die benadering van hierdie studie is deur middel van 'n praktiese voorbeeld getoets en dit het belowende resultate opgelewer.

LIST OF ABBREVIATIONS

DBMS	Database Management System
psql	PostgreSQL
DBA	Database Administrator
ANSI	American National Standards Institute
SPARC	Standards Planning and Requirements Committee
FTP	File Transfer Protocol

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION.....	2
1.1 Overview of Current Literature.....	1
1.2 Problem Statement.....	2
1.2.1 Context of the Problem	2
1.2.2 Research Problem	3
1.3 Research Objective and Questions.....	3
1.4 Approach.....	4
1.5 Research Contribution.....	5
CHAPTER 2 DIGITAL FORENSICS.....	6
2.1 Introduction	6
2.2 Digital Forensics Defined.....	7
2.3 Real World Incidents	8
2.3.1 Bank Hacker and the Swift Exploit.....	8
2.3.2 BTK Killer and the Investigation Success	9
2.3.3 Operation Oplan Bojinka and the Concern	10
2.4 The Digital Forensic Phases.....	11
2.4.1 Identification	11
2.4.2 Collection.....	12
2.4.3 Transportation	14
2.4.4 Storage	14
2.4.5 Analysis.....	15
2.4.6 Reconstruction	15
2.4.7 Presentation.....	16
2.4.8 Destruction.....	16
2.5 Digital Forensic Tools	16
2.6 Existing Digital Forensic Fields.....	18
2.6.1 File System Forensics	18
2.6.2 Network Forensics	20
2.6.3 Mobile Device Forensics.....	20
2.6.4 Internet and Email Forensics.....	21

2.7	Conclusion.....	22
CHAPTER 3 DATABASES.....		24
3.1	Introduction	24
3.2	A Brief History Of Databases.....	24
3.3	DBMS Architecture.....	26
3.3.1	The ANSI SPARC Model	26
3.3.2	Components of the DBMS	30
3.3.3	Execution Steps of the DBMS	31
3.4	The Relational Data Model.....	32
3.4.1	Relational Algebra	32
3.4.2	Relational Data Model Structure.....	34
3.4.3	Structured Query Language	35
3.5	Database Design.....	37
3.5.1	Introduction to Physical Database Design.....	37
3.5.2	Indexing	38
3.5.3	Query Optimisation	38
3.5.4	Decomposition	40
3.5.5	Normalisation.....	40
3.6	Database Security.....	41
3.6.1	Background	41
3.6.2	Threats.....	42
3.6.3	Access Control	42
3.6.4	Data Encryption	43
3.6.5	Statistical Database Security	44
3.7	Database Trends.....	44
3.8	Conclusion.....	45
CHAPTER 4 DATABASE FORENSICS		46
4.1	Introduction	46
4.2	Dimensions of Database Forensics	48
4.2.1	Damaged or Destroyed Databases.....	49
4.2.2	Compromised Databases	51
4.2.3	Modified Databases.....	53
4.2.4	Orthogonality of Dimensions in Database Forensics	53

4.3	Database Forensic Tools	55
4.4	Four Abstract Layers of the Database Management System	56
4.4.1	Background	57
4.4.2	Why are abstract layers useful in a forensic investigation?.....	58
4.4.3	Abstract Layers of the Database Management System	58
4.4.4	Hierarchy of Abstract Layers	63
4.4.5	Boundaries Between Abstract Layers	65
4.5	Conclusion.....	67
 CHAPTER 5 DATABASE DATA MODEL FORENSICS.....		68
5.1	Background.....	68
5.2	Forensic Environment With Either a Clean or Found Data Model.....	71
5.2.1	Arguments for a Clean Data Model Environment.....	71
5.2.2	Arguments for a Found Data Model Environment.....	74
5.3	How to Achieve Various States of the Data Model.....	76
5.3.1	Clean Environment	76
5.3.2	Found Environment.....	82
5.4	Data Model Forensics in Context With the Other Abstract Layers of the DBMS	86
5.5	Final Comments.....	86
5.6	Conclusion.....	87
 CHAPTER 6 DATABASE DATA DICTIONARY FORENSICS.....		88
6.1	Introduction	88
6.2	What is the Data Dictionary of a DBMS?	88
6.3	Data Dictionary Attacks and Alterations	90
6.3.1	Compromised Databases	91
6.3.2	DBMS User Alterations	92
6.4	Selecting a Forensic Environment for the Data Dictionary	93
6.4.1	Arguments for a Clean Data Dictionary.....	93
6.4.2	Arguments for a Found Data Dictionary	94
6.5	Methods to Convert the Data Dictionary into a Forensic State	95
6.5.1	Methods to Achieve a Clean Environment.....	95
6.5.2	Methods to Achieve a Found Environment.....	97

6.6	Conclusion.....	98
CHAPTER 7 DATABASE APPLICATION SCHEMA FORENSICS.....		99
7.1	Introduction	99
7.2	Argument for Application Schema Forensics	100
7.2.1	Damaging the Application Schema.....	101
7.2.2	Application Schema Alterations to Deliver Wrong Results	103
7.2.3	Database Behaviour Alterations.....	106
7.3	Selecting an Application Schema Forensic Environment	107
7.3.1	Arguments for a Clean Environment	108
7.3.2	Arguments for a Found Environment.....	110
7.4	Methods to Deal with Application Schema Forensics	111
7.4.1	Clean Application Schema Environment	111
7.4.2	Found Application Schema Environment	113
7.5	Conclusion.....	114
CHAPTER 8 PRACTICAL APPROACH AND RESULTS.....		116
8.1	Introduction	116
8.2	Approach.....	117
8.2.1	The DBMS Forensic Matrix.....	118
8.2.2	Risk Profiling	121
8.2.3	Evidence Identification Process for DBMS	123
8.3	Practical Example	125
8.3.1	Compromising the DBMS and Setting Up a Test Environment	126
8.3.2	Risk Profiling	128
8.3.3	Select Forensic Scenario	129
8.3.4	Method to Convert Environment.....	130
8.3.5	Implementing Methods to Convert Environment.....	130
8.3.6	Results.....	135
8.4	Conclusion.....	138
CHAPTER 9 CONCLUSION.....		139
REFERENCES		142

CHAPTER 1 INTRODUCTION

As our lives become increasingly dependent on technology the storage of data in databases is increasing. The human race is becoming so reliant on technology that our daily whereabouts are stored in our cell phone company's database; human records are stored in the Department of Home Affairs' database; our private banking information is stored in databases; the balance of our banking account is stored in a database; databases hold medical history of individuals etc.

The phenomenon of criminal activity or more philosophically stated, unacceptable behaviour, has been partner to the human race for many centuries. As technology becomes more interwoven into our society, criminal activities on digital devices are increasing. A forensic investigation is required to collect evidence to determine what has happened on a crime scene and who is responsible for the crime. The database is not free from such criminal activity and forensic investigations will be required on databases. In fact, the information on databases is sensitive and could be connected to a person's private information or finances. There is no reason why the database technology will not be victim to the same criminal activity that is seen on other digital devices. However, the field of database forensics is still in its infancy.

1.1 OVERVIEW OF CURRENT LITERATURE

A basis for database forensics has been suggested by identifying the similarities between file system forensics and database forensics due to the similarities that exist between file systems and databases [23]. The majority of work done on database forensics was done in corporate environments [88,130] where clients asked professionals to investigate their databases for possible attacks. This information is very specific towards the database management system installed on the server (MSSQL, ORACLE, etc.) and interlinked with database security [8,79,80,81,82,83,84,85,88,130,131]. The information is also documented separately for some unique DBMS, but little or no attempts are made to document database forensics as a structured methodical field. Information is available on security flaws in database management systems that can be exploited to receive unauthorised information or launch attacks on databases [132]. The flaws in the security of well-known database management systems like Oracle, DB2, Sybase, MSSQL, etc. are

illustrated. This information is useful in order to gain knowledge on the type of attacks that will typically be investigated. In other literature, data mining techniques have been proposed to simplify forensic investigations on large datasets [89].

The field of database forensics has not been explored extensively. Some database forensics literature exists on a case-by-case basis but very little studies have attempted to solve the challenges which the field of database forensics presents as a whole. Although the notion of compromised databases has been published in literature before [23], an in-depth study has not yet been done on how to solve the challenges that compromised DBMSs present to a forensic investigator. This lack of literature in the field of database forensics and the lack of literature about forensic investigations on compromised databases introduces a research gap.

1.2 PROBLEM STATEMENT

1.2.1 Context of the Problem

A consequence of the lack of literature in the field of database forensics is that compromised DBMSs are not well-known or understood. In this study a compromised DBMS refers to a DBMS that has been manipulated to deliver false or misleading results. Olivier [23] pointed out that the DBMS is a software program that consists of code, metadata and data. The software can be changed in an endless variety of ways to manipulate the results of the DBMS. Additionally, the metadata of the DBMS could be changed within the rules of the DBMS to deliver incorrect results or even hide data. The challenge posed by compromised DBMSs is that the DBMS itself is often used as a forensic tool to extract evidence from its data structures but if the DBMS is compromised the results cannot be trusted and therefore the results cannot be used as evidence in a court of law.

The ANSI/SPARC model [100] divided the DBMS into four layers which illustrated that the DBMS consisted of code, metadata and data. The four layers are the data model layer, data dictionary layer, application schema layer and the application data layer. A DBMS

can be compromised on one of the top three layers which we shall call “the extensional layers”. The extensional layers are singled out because they have the potential to manipulate the results of the DBMS. If the extensional layers of the DBMS have been compromised, it disqualifies the evidence extracted from the DBMS.

The integrity of a DBMS layer can be restored by converting a layer into a clean state. A clean state refers to a forensic state of an abstract layer of the DBMS where the layer has been cleaned up by making use of a reliable method that does not influence the integrity of the results. However, to convert a layer to a clean state is not always easy or possible to achieve. Therefore, the DBMS also needs to have some layers that remain in a state in which they were found where the layer has not been cleaned up. Therefore, the forensic investigator might often need a combination of clean and existing layers to conduct a forensic investigation on a compromised DBMS.

1.2.2 Research Problem

The problem to be addressed is how to construct the appropriate combination of clean and existing layers for a forensic investigation on a compromised DBMS while ensuring the integrity of the forensic results.

1.3 RESEARCH OBJECTIVE AND QUESTIONS

The objective of this study is to illustrate empirically how a forensic investigation can be conducted on a compromised DBMS by combining various forensic states of the layers of the DBMS while ensuring the integrity of the evidence in the DBMS. Since criminal activity, negligence, malicious intent etc. on digital devices, including databases, is increasing it is vital to conduct forensic investigations to identify what has happened. It therefore poses a problem that so little literature exists on database forensics, especially due to the fact that databases hold so much sensitive information. Compromised DBMSs introduce the problem that incorrect evidence may be delivered in court. It is therefore useful to conduct a study that contributes to the field of database forensics and that can solve the challenges posed by compromised DBMSs.

In addition to the problem statement, the following research questions were considered and answered in this study:

- How can the code or metadata of the DBMS be modified in order to compromise the DBMS to deliver incorrect evidence?
- How can an approach be found that forensic investigators could use on a variety of DBMS-types which might have been compromised?
- Can the challenges that compromised DBMSs present be identified and solved in a relatively young field of database forensics?

1.4 APPROACH

The approach of this study consists of three central sections. Firstly, the PostgreSQL DBMS is divided into four abstract layers according to the ANSI/SPARC model where these four abstract layers could be applied to any relational database. Secondly, this study illustrates how the layers can be prepared to be either in a found or clean forensic state. Thirdly, the clean and forensic states are introduced in this study in order to find a way to restore the integrity of a compromised DBMS. Lastly, the study experimentally combines the prepared layers of the DBMS according to the forensic scenario so that a forensic investigation could be conducted on a compromised DBMS.

The study commences with background information on the subjects of databases, digital forensics and database forensics respectively to give the reader an overview of the literature that already exists in these relevant fields. The study then discusses the four abstract layers of the DBMS and explains how the layers could influence one another. The clean and found environments are introduced due to the fact that the DBMS is different to technologies where digital forensics has already been researched. The study then discusses each of the extensional abstract layers individually, and also discusses how and why an abstract layer can be converted to a clean or found state.

A discussion of each extensional layer is required to understand how unique each layer of the DBMS is and how these layers could be combined in a way that enables a forensic

investigator to conduct a forensic investigation on a compromised DBMS. It is illustrated that each layer is unique and could be corrupted in various ways. Therefore, each layer must be studied individually in a forensic context before all four layers are considered collectively. A forensic study is conducted on each abstract layer of the DBMS that has the potential to influence other layers to deliver incorrect results. Ultimately, the DBMS will be used as a forensic tool to extract evidence from its own encrypted data and data structures. Therefore, the last chapter shall illustrate how a forensic investigator can prepare a trustworthy forensic environment where a forensic investigation could be conducted on an entire DBMS by constructing a combination of the appropriate forensic states of the abstract layers.

1.5 RESEARCH CONTRIBUTION

This study contributes to the young field of database forensics by providing literature that will assist readers in understanding the field of database forensics better. The importance of considering compromised DBMSs in a forensic investigation was highlighted and an approach was provided to identify evidence on a compromised DBMS. The same approach could be used to identify evidence on DBMSs that were not compromised.

If experts in the field of database forensics start to seriously consider the fact that a DBMS might be compromised, it could bring about a mind-shift in the way in which DBMS forensic investigations are approached at the moment. For example, database log files cannot be analysed blindly for evidence without considering that the code that generates the log files might be compromised. The ultimate goal of forensics is to accurately determine what has happened. The author believes that this study will help to make the field of database forensics more accurate by eliminating unawareness towards compromised DBMSs and empirically illustrating how such an investigation can be conducted.

CHAPTER 2 DIGITAL FORENSICS

The background of digital forensics will be researched in this chapter to familiarise the reader with the work that has been done in the broader field of digital forensics. The field of digital forensics is a well-researched field and important aspects of digital forensic literature are presented to the reader in this chapter. Elements of the work accomplished in digital forensics might be extended to database forensics.

2.1 INTRODUCTION

Computer devices and networks have increasingly become part of normal human life and inevitably became part of virtually all human activities. Human activities are made faster, safer and more exciting by technology. We use devices to shop, communicate, create intellectual property, plan trips, perform business transactions [3] etc. The combination of computer devices and networks enable users to make continuous use of these digital devices like personal digital assistants (PDAs), cell phones, smartphones, wireless connection devices and other devices with connectivity to the Internet. As computers become so broadly involved in human activities they are increasingly connected to incidents of crime. Therefore, it should be no surprise that the digital evidence brought before courts are increasing dramatically [4].

Enterprises are more than ever before reliant on digital devices and the data on devices. If the digital devices or data of an enterprise were compromised, it would have vast negative ramifications on the reputation and general functions of the enterprise. Such a breach could be disastrous to an enterprise like when Sony's PlayStation Network was hacked in April 2011, and customer details and possibly customer credit card information were exposed by hackers [14]. Sony contacted about 70 million customers to warn them about a breach of their names, addresses, birth dates, passwords, security questions and possibly credit card information. Sony was sued for \$1 billion, lost customers and their reputation suffered a great deal. This example illustrates how much large enterprises rely on digital devices and the data stored on them. It has the potential to build an enterprise into a world leader or be disastrous to it [9].

With the increase of digital crimes and fraudulent behaviour on digital devices, we need to develop, maintain and update the forensic science to determine what has occurred during a digital crime. With the fast changing technology revolution it is difficult for law enforcement and computer security professionals to stay one step ahead of digital savvy criminals. Therefore, it is important that the digital forensics field keeps on developing and keep up with the digital crimes being committed. Many computer professionals still perceive digital forensics as a type of incident response when a digital breach or crime has occurred, but the scope of digital forensics stretches wider than that. When a digital crime has occurred, it is critical to understand how to deal with the evidence in order to fix the vulnerability and possibly prosecute guilty parties. The process of finding and understanding digital evidence is the core objective of digital forensics [13].

2.2 DIGITAL FORENSICS DEFINED

Numerous experts and authors have attempted to define digital forensics based on their knowledge and experience. The result is a spectrum of definitions that share some common elements, but do differ from one another. Vacca [12] described digital forensics as the collection, preservation, analysis and presentation of digital-related evidence. Vacca uses the forensic process within his definition which usually includes these four processes. Ray [15] used various sources of digital forensic definitions to find common denominators from which to build a new definition of digital forensics. He concluded that digital forensics is the need for a forensic process to maximize the evidentiary weight of the resulting electronic evidence. Ray describes a scientific definition of digital forensics which focuses on the final result of the digital forensic process.

Pollit [16] provided a definition of digital forensics almost two decades ago which states that “digital forensics is the application of science and engineering to the legal problem of digital evidence. It is a synthesis of science and law. At one extreme is the pure science of ones and zeros. At this level the laws of physics and mathematics rule. At the other extreme is the courtroom.” Pollit’s definition paints a very realistic picture of what digital forensics is. At the one extreme is the physical binary data that needs to be searched for digital evidence. At the other extreme is a courtroom that needs to consider this digital

evidence. There is a considerable distance between the physical binary data, and the courtroom where human interpretation and human emotions are involved. The evidence needs to be found in the binary data in a way that ensures that the integrity of the evidence was not lost. Additionally, the digital evidence must be presented in a way that is understandable in court.

2.3 REAL WORLD INCIDENTS

This section will describe various examples of real world incidents where digital forensics was required. The aim of this section is to give the reader a realistic view of what type of investigations are required in digital forensics and illustrate to the user that high profile crimes could be solved by digital forensics.

2.3.1 Bank Hacker and the Swift Exploit

On January 25, 2003 the system administrator of a regional bank thought it will enhance the rule of the bank's Cisco router by setting the first rule of the router to *IP permit ANY ANY* [3]. This change removed all access restrictions from the router that was used as an Internet demilitarizing zone (DMZ). A month later the system administrator realised that the Internet connection became very slow. An investigation by the system administrator revealed that large amounts of data were being transferred to and from the FTP server of the DMZ router. The FTP server permitted anonymous FTP with read and write access. Commonly this kind of exposure is exploited by software pirates and media lovers to store movies and illegal software programs. The administrator found movies on the server such as *Tomb Raider* and *Star Wars*.

Usually the investigation for an administrator will end here. The access settings of the router will be corrected and anonymous FTP access will be stopped. However, because this exploit occurred in a banking environment with very sensitive data, a further investigation was required to determine whether the sensitive information was accessed. Further investigation revealed the following exploits. The web server and FTP server were configured to have the same root directory. This means that files and directories that were accessible via the FTP server were also accessible via the web server. The FTP server did

not allow files to be executed, but now the files were executed by making use of the web server. Now files could be uploaded and executed on the server. The FTP log was investigated and revealed that several Active Server Pages (ASPs) were executed on the server. At this point it was clear that the server was completely compromised from the Internet.

This example shows how swiftly and severe an exposure can be exploited via the Internet. This is a very general digital forensic case and illustrates how networks can be used to commit digital crimes, cause damage to organisations' networks and data, or view private information. This example also illustrates that digital forensic investigations are not always used in court cases, but sometimes they are used to merely determine what has happened.

2.3.2 BTK Killer and the Investigation Success

The digital forensic investigation that led to the arrest of the BTK serial killer is probably one of the most famous cases that was solved by making use of a simple digital forensic investigation. The BTK killer was responsible for the deaths of ten people in Wichita, Kansas, USA [6]. Several years later a series of communication was received from the BTK killer and one of the communication methods used was a floppy disk. The police authorities found information in the metadata of a deleted Microsoft Word document on the floppy. The metadata included a name "Dennis" as well as a link to the Lutheran Church. When the police searched "Dennis Lutheran Church Wichita" on the Internet, they were able to identify a suspect. Upon further DNA tests Dennis Rader was positively identified as the serial killer and Rader himself consequently confessed to the killings. Rader was sentenced to a minimum of 175 years in prison.

This example illustrates what a vital part digital forensics is of investigations because of the fact that digital devices have become such an important part of people's everyday life. People (including criminals) use computers, networks and other electronic devices to communicate, to store vital data on and to process various everyday tasks. In this example digital forensics did not provide the substantial evidence of who specifically the suspect is,

but it provided the hint which led the investigators to the suspect. It seems that investigators are still just after the story, but the computer can be a witness now [7].

2.3.3 Operation Oplan Bojinka and the Concern

During February 1993 a minibus filled with 500 kg of explosives was driven into the parking area beneath the World Trade Centre Towers in New York [5]. The explosion claimed the lives of six people, injured around a thousand people and was estimated to have caused damage worth \$300 million. A number of suspects were sought after by the U.S. investigators. Amongst the list of suspects the names of Ramzi Ahmed Yousef and Abdul Hakim Murad appeared. Two years later a fire broke out in a suspected terrorist safe house in Manila in the Philippines. In the safe house it is believed that the two suspects (Yousef and Murad) prepared explosive devices. Yousef fled the Philippines and Murad returned to the apartment to remove evidence of their activities. The police arrived at the apartment and arrested Murad. The police searched the apartment and recovered evidence that ranged from explosive devices to a laptop computer.

A forensic investigation of the laptop both amazed and apprehended the investigators. The laptop belonged to Yousef, and contained information of both past and future terrorist activities. This information included airline flight schedules, detailed plans pertaining to a past attack on a Philippines airline and details of project Oplan Bojinka. The details of project Oplan Bojinka was particularly disturbing. The project involved the kidnapping of commercial airliners flying to the USA and using the airliners to attack key targets. The mentioned targets included the FBI headquarters and CIA headquarters. The success of the investigation was the recovery of information pertaining terrorist attacks, but out of a digital forensic investigation perspective a concern was raised. The majority of information on the laptop was encrypted and could not be investigated. The concern was raised that law enforcement needed to keep up with technology and that the widespread use of robust encryption ultimately will devastate the ability to fight crime and terrorism. Digital forensics should enhance as technology enhances.

2.4 THE DIGITAL FORENSIC PHASES

In order to complete a forensic investigation that was required in the three examples discussed in the previous section, the forensic investigators were required to work through several phases. Traditional digital forensics can be divided into four phases, namely the collection, preservation, analysis and presentation phases. These phases are used by normal forensics as well as digital forensics. These four digital forensic phases fulfil the needs of reactive digital forensics where an incident has already occurred and the scene of the incident needs to be searched for evidence.

In this section a more complete process is discussed, as described by Cohen [18] who considers the identification, collection, storage, analysis, reconstruction, presentation and destruction of digital forensic evidence. Various digital forensic process models (staircase model, evidence flow model and subphase model) exist, as depicted in [24]. However, we will focus on the model proposed by Cohen [18] due to its potential relevance to the field of databases.

2.4.1 Identification

Digital devices need to be searched to identify relevant evidence. It is common that an enormous amount of potential evidence is identified and that a large amount of evidence is never discovered. To illustrate that it is difficult to identify all evidence, consider a situation where a network device has done some malicious activity that needs to be investigated [18]. The evidence can exist on a remote device of which we do not have knowledge and which is located on the other side of earth. To identify this evidence might take a long time and by the time the evidence is discovered, it can either be destroyed or the evidence may not be relevant anymore.

On the other extreme a huge amount of evidence may be identified when we take into consideration, for example, that a single executed task on a digital device can trigger the processors and memory of that digital device to interact with files or with the file system, and produce logs and audit trails. The identification process may also entail reducing the amount of data to be investigated [24].

2.4.2 Collection

The collection phase, also frequently called the acquisition phase, is the phase when digital media are collected to be examined [18]. The digital media include physical hard drives, optical media, storage cards of digital cameras, mobile phones, chips from embedded devices, document files [15] etc. The manner in which the evidence is collected is important, because the collected evidence should preserve its integrity to be considered for use in court. The chain of custody should be preserved in order to keep a paper trail of the conditions under which the evidence was collected and preserved.

Evidence that is collected is driven by what is identified. The chunks of data or logs that have been identified during the identification phase are used to collect the actual evidence that could be used in court. This is not an easy task and we need to consider that many systems today cannot afford to pause their usual functions in order to make time for a forensic investigation. Consider the situation where an Internet service provider (ISP) needs to pause one of its connection critical servers for an investigation or where an Internet online store needs to make one of its purchase critical servers available for an investigation. This can cost the company that assists in the investigation a great sum of money. Therefore, the method of evidence collection needs to be thought through.

The methods that are used to collect evidence have been the centre of debate in the digital forensics community for years and will probably remain a point of discussion for years to come. To complicate the matter further, the collection methodology might also be influenced by the client or an employer [19]. However, the debate about collection methodology mainly contends between live evidence collection opposed to static evidence collection.

Imagine a forensics investigation done at a theft crime scene (maybe an investigation correctly or incorrectly done at a crime scene in many a television series or movie) where evidence is collected. The investigators arrive on the crime scene after the crime event has occurred. The investigators will search the area to identify evidence. Once evidence is identified it has to be collected. Can we consider this collection as live forensics or post-

mortem forensics? If a hair sample is extracted from the crime scene to be examined in a laboratory, the hair sample is collected in a way that does not compromise the sample in any way. This is a typical post-mortem forensics example. Just like the hair sample is extracted from the crime scene, we may extract a digital device from the crime scene for a post-mortem forensic investigation. The big difference lies in the fact that our method of collection might influence the evidence. A hair sample for instance cannot be influenced when picked up with tweezers and stored correctly, but when a digital device is shut down (pulled the plug) when it is collected it might influence the evidence. This is the big debate concerning live versus post-mortem forensics.

Digital devices frequently keep data in their volatile memory which requires an electronic source at all times to maintain the state of the memory. If the device loses the electronic power source the memory is lost. This type of technology is used because it is much faster than non-volatile memories. Therefore, evidence might be lost when the digital device is plugged out [107]. Another fact is that in some situations it might just be better to conduct a live forensic analysis. Despite this fact, the post-mortem way of conducting a forensic analysis has been accepted as the best practice methodology, mainly because a live forensic analysis might influence the evidence even more than pulling the plug does [133]. The hard drive, file time stamps, registry keys, swap files, memory and the complete md5 hash of the evidence are some of the pieces of evidence that might be influenced by a live forensic analysis [18]. This poses a great case for post-mortem digital forensics.

There are situations where post-mortem digital forensics becomes very difficult or almost impossible and live digital forensic analysis is the only option or a much more effective option [106]. Many organisations do not have one location where all their servers are held. The servers might be located across a city, country or continent. This makes it a daunting (or financially impractical) task for a forensic investigator to collect all the evidence on location. Not even to mention that the organisation relies on some of these servers to make money and that removing these servers (for even just a while) will cost the organisation a lot of money.

Another technology that is making post-mortem forensics infeasible is encrypted file systems [134]. Mirroring this digital device might only deliver unreadable chunks of encrypted data that cannot be understood by the investigator. A live analysis on an encrypted file system might give the investigator access to opened files or readable data that are still in the memory in a decrypted state. In these scenarios there is a case for live digital forensics. Most recent trends in technology advancement would make it seem that this method of digital forensic evidence collection becomes more popular due to the nature of digital devices to become more and more physical location independent than ever before [15].

2.4.3 Transportation

Digital evidence sometimes needs to be transported from the crime scene to be kept in a secure location [12]. The transport methods range from physically removing the digital device from the crime scene and transporting it by vehicle to the secure location to copying the evidence over a network to the secure location by ensuring that the digital evidence preserves its integrity [18]. Copies of the evidence are usually kept in a secure location in order for the evidence to be referenced anytime during legal proceedings. Evidence is increasingly being transported electronically from place to place and the smallest error can cause the evidence to arrive incorrectly at the secure location. A chain of custody must be kept to report on how the evidence has been transported and witnesses must be able to testify how the integrity of the evidence has been preserved during transportation.

2.4.4 Storage

The digital evidence must be stored in the right conditions [24]. Depending on the media these conditions can be the correct temperature range, correct humidity range, correct power supply etc. The digital evidence is required to be stored and maintained for the remainder of the trail until the time that the evidence is no longer required. Many different sorts of things can go wrong during storage, such as evidence decaying over time, being physically harmed, condition changes that influence the stored evidence like fires, floods, etc.

2.4.5 Analysis

The analyst responsible for interpreting the evidence should consider all possible explanations to determine what actually has occurred and how certain he is of his assumptions [18]. It is a common occurrence that supposed experts draw conclusions which are not justified. An analyst will typically strive to report in court that according to evidence available, it appears X did Y producing Z, where X is a person or a program and Y is the action which produced the evidence Z. It also helps if all alternative explanations that could have produced Z are explored and proved inconsistent. In order to prove that alternative explanations are inconsistent, seemingly useless evidence might prove to be very useful. For example, a seemingly irrelevant log file may prove that the digital device was not shut down and that might disprove a possible alternative explanation.

Powerful and sophisticated tools have been developed to analyse digital and storage devices, and to extract potential evidence from these digital media. Some digital forensic suites (EnCase, iLook, FTK etc.) have been developed to revolutionise the way in which evidence is analysed on digital media [106]. With graphical user interfaces the task of analysing digital media is made simpler by enabling the analyst to extract potential evidence with a software single tool. Examples of potential evidence that can be extracted by these tools include recovering deleted files, searching files and slack space, extracting and processing email, parsing log files, analysing the Windows registry, performing metadata and timing analyses, and creating reports [17].

2.4.6 Reconstruction

Crime reconstruction is the process of gaining a comprehensive understanding by making use of available evidence. Forensic examiners perform a reconstruction to determine how a particular system, device or application works in order to better understand a piece of digital evidence or how the crime could have occurred [18]. The examiner might have to create an exact replica of the system in order to perform a reconstruction. Reconstruction is often used by forensic examiners when the crime has occurred a long time ago and the only digital evidence might not be available anymore. It is important to consider if the hardware or software is exactly the same model or version when conducting the reconstruction. If the

hardware or software of the reconstruction is not the same as the original hardware or software, it should be proved that the original hardware or software would have delivered the same result as the reconstructed hardware or software.

2.4.7 Presentation

Jurors or judges often have little knowledge about the particular technology related to the court case and this poses a challenge to the presenter of evidence to make his findings comprehensible to the court. The digital evidence presentation phase includes the summarisation of the conclusion drawn during the analysis phase as well as an explanation of the collection and examination techniques used [19]. Evidence is usually presented in the form of expert reports, depositions or testimonies.

2.4.8 Destruction

Courts often order evidence and related material to be destroyed or returned after it is not needed for the purposes of the court anymore. This applies to trade secrets, confidential patents, client-related information, copyrighted works and information that organisations usually get rid of but was preserved for legal purposes.

2.5 DIGITAL FORENSIC TOOLS

This section will discuss digital forensic tools because they are often required to assist a forensic investigator through the phases of digital forensics discussed in the previous section. Digital evidence is often latent in nature, meaning that it is present, but not apparent. Therefore, digital forensic tools are regularly required to assist an investigator in gathering evidence. The legal system mainly allows forensic tools that have been properly applied by experts who know how to use the tools properly [18]. When making use of digital forensic tools, the investigator needs to understand what function is carried out by the forensic tool. He thus examines the result of the tool for anomalies before declaring the results of the tool to be precise and accurate. If inconsistent results are delivered by the expert who has made use of a forensic tool and the expert's conclusions are proven to be incorrect, the expert might be excluded from the remainder of the legal process and the reputation of the forensic tool could be harmed.

EnCase is a forensic tool that has built up a good reputation. EnCase first appeared on the market in 1998 at a time when most examiners made use of the DOS command prompt to conduct most of their forensic investigations. Many never imagined that the software tool will become the leading digital forensic tool in 2000 [20]. EnCase was unique in the sense that it mounted a bit-stream of forensic images as virtual read-only devices. EnCase then reconstructed the file system by reading data from the forensic images, thus ensuring not to alter data on the suspect machine. EnCase caused many experts to convert from command-line evidence searching to a forensic tool with a GUI.

Many forensic tools exist today, both GUI and command line based. A list is provided by [19] of more than 300 forensic tools available today. The tools are divided into the sections that include the following:

- Slack space and data recovery tools assist in the recovery of deleted files or data, and the recovery of file fragments located in slack space on a file system supported by Windows. Ontrack and DriveSpy are both examples of slack space recovery tools.
- Data recovery tools may recover files from many sources including PDA, mobile devices, cameras and disk drives. Device Seizure aids in forensically recovering messages, photos, call logs and other data from cell phones, smartphones and PDAs. Further data recovery tools include Directory Snoop and Forensic Sorter.
- File integrity checkers help investigators prove that a copied file can be considered to not be altered. These tools make quick analysis of systems to ensure that the state of the system is the same.
- Disk imaging tools create bit-map images of storage devices or other media. SnapBack DatArrest obtains images of different operating systems and makes backups of data on a hard disk.
- Partition managers like PartImage can store partitions to an image file and write the image of the partition onto another source to be collected as potential evidence.
- Several Linux or Unix tools can be used to assist in forensic investigations. Ltools make use of several command line tools which could be executed on Windows

systems to be used in a similar fashion than normal Linux tools. Similar tools include Mtools and Tctutils.

- Password recovery tools are tools that execute a huge list of regularly used passwords and dictionary words. Sometimes the plain text needs to be hashed to recover a password. Most password recovery tools (like @Stake, Decryption Collection Enterprise, AIM Password Decoder) are used for password auditing purposes.
- Multipurpose tools like Maresware may cover a large range of features to assist in forensic investigations. The list of functionalities is exhaustive and can be found in [19].
- Toolkits offer a compilation of forensic tools in one software program. Examples of toolkits include NTI Tools, Stealth Suite, Data Elimination Suite, TextSearch Suite, SafeBack, R-Studio, EnCase, Forensic Toolkit (FTK) etc.

2.6 EXISTING DIGITAL FORENSIC FIELDS

Although the field of database forensics has not received a lot of attention over recent years, other subsections of digital forensics have enjoyed more attention. Some existing fields of digital forensics potentially have common characteristics to databases and database forensics might be solved in a similar way. Therefore, this section will focus on file system forensics, network forensics, mobile device forensics, and Internet and email forensics.

2.6.1 File System Forensics

This study will ultimately deal with database forensics which is closely related to file system forensics [23]. File systems rely heavily on metadata to organise the data stored in memory, just like database management systems do. There are three components to proper file system forensic analysis: (1) to understand the file system; (2) to understand the artefacts within the operating system and how to find them and interpret them, (3) and to make use of proper forensic software [20,21].

An operating system (OS) like Windows consists of various structures, features and artefacts that can prove to be very useful in a forensic investigation. In modern Windows versions the most noticeable of these structures can be defined as the NTFS (new technology file system), the registry, files, slack space, swap file, unallocated clusters, unused partitions, hidden partitions etc. Although there are many file systems out there such as FAT, Ext2, Ext3, UFS1, UFS2, we will briefly mention the NTFS file system due to its much superior number of uses in the world.

The NTFS is significantly different from its predecessor, the FAT file system. The NTFS makes use of several metadata files to keep track of files and folders in memory. The master file table (MFT) is the heart of the NTFS because it contains all information about all files and directories [22]. Every file and directory has at least one entry in the table. Each entry in the MFT describes the file's name, content and address as well as other configurable attributes. The MFT also keeps record of the address of a file on the storage device. The location of the MFT is specified in the boot sector of the operating system. In a similar fashion as database management systems, the NTFS provides the capability to encrypt its data. The \$DATA attribute of a MFT record can be encrypted by making use of asymmetric encryption techniques. As discussed in previous sections, it is difficult to do a post-mortem analysis on encrypted data, because the data itself is in an unreadable format on the disk. A live analysis is regularly required to view encrypted data.

Some analysis techniques to analyse the file system include the following:

- Process the boot sector in order to find the location of the MFT. The MFT can be analysed to reveal the structure of the file system.
- The number of sectors in the file system should be compared to the actual size of the storage device to make sure there is no slack space or unused space after the file system. This slack space might be used to hide data.
- The end of the MFT file can be used to hide data, but this is a risky place to hide data because it can easily be overwritten by normal operating system activity.

- It can also be useful to investigate the attributes of the file system metadata files, because additional attribute names could be assigned to the metadata files to hide data [23].
- Create a valid mirror of the drive by modifying system files to prevent the operating system from accessing system components on the evidentiary drive [24].

2.6.2 Network Forensics

Hardly a day goes by without news of a major network intrusion of a major company or government [121]. The network intrusion industry has grown extensively in recent times and has also become a critical matter of national security. Success stories of network forensics include:

- Determining where a stolen laptop was last traced by interconnected WAN (or Hotspot) devices in a hospital [122];
- tracing a corporate user who pirates films by tracing the IP address from switches of other networking devices; and
- determining the extent of a brute force attack on a hacked government server by analysing SSH logs.

These abovementioned examples provide an illustration of the reality of what network forensics entails. The field of network forensics might require a unique solution to each problem, but requires a structured approach to investigations, as is the case in the greater digital forensics field. The techniques which are commonly used in network forensics include network packet analysis, statistical flow analysis, intrusion detecting, intrusion analysis, event log analysis, network device evidence gathering techniques and web proxy analysis. More information about these various network forensic analysis methods can be found in [122].

2.6.3 Mobile Device Forensics

Many people question how they have ever managed without a cell phone, but it is less than two decades ago that the cell phone has become available. Most people have a cell phone or PDA of some sort today. Cell phones and PDAs have evolved into smartphones and

tablet PCs. Similarly to databases, these mobile devices have become such an interconnected part of people's lives that many crimes (digital and non-digital) cannot be fully investigated without investigating mobile devices linked to the crime. These mobile devices may hold communication information, location information and could even be attacked to retrieve personal information. A mobile device or PDA now has roughly the same computing power of a computer manufactured within the last five years [123].

Mobile forensics holds various new challenges in a forensic context which include:

- Frequent change of operating systems, interface methods, hardware standards and storage technologies;
- various different mobile device platforms; and
- wireless technologies which are used to communicate.

In the previous section file system forensics was discussed. Mobile devices might have different file systems on which to do forensic investigations, depending on the make and model of the phone. Handbooks are already available on Google's Android operating system [124] and Apple's IOS operating systems [125].

Evidence found on a cell phone and mobile devices include pictures, videos, contacts, calendars, deleted text messages, location information, voice memos, a browser history, chat sessions and documents [122]. Call details records (CDRs) are generally used by mobile operators to bill cell phone network users, but they may also contain information valuable to examiners such as call times, calling number, caller number, call time length and cell phone towers utilised. Various mobile device forensic tools such as BitPim, Oxygen, AccessData's MPE also exist which could assist an investigator during an examination. More information on mobile forensic tools could be found in [122].

2.6.4 Internet and Email Forensics

The Internet is a type of network and could therefore be categorised as network forensics. However, the Internet and email technologies have established their own research field due to the high user volumes of these technologies. The Internet and email technologies have

become targets for various attacks on end users who make use of these technologies. These attacks include spamming, phishing, viruses, worms, and the convergence of viruses and spam.

A report in 2004 stated that 73% of all emails are spam and computer viruses were identified in 6% of all emails [126]. Furthermore, 4 500 000 phishing attacks were measured in the second half of the 2004 calendar year. Email forensics entails analysing message headers and tracking spammers. Internet forensics entails analysing web browser usage, analysing web servers for attacks, analysing websites for attacks, analysing file contents and determining the location of an Internet user.

This section has discussed various existing fields of digital forensics. The various existing digital forensic fields may hold some useful approaches that could be used in a database forensic context. For example, the structure of a database might be similar to a file system at a very high level. The database connects to servers via a network which might make network forensics relevant. Mobile users access content in databases via the Internet which might bring mobile and internet forensic approaches into consideration.

2.7 CONCLUSION

In this chapter the reader has been familiarised with the field of digital forensics and the work that has been done in this field. The chapter has focused on the digital forensic phases and what it entails to conduct a forensic investigation. Forensic investigations are not completed after the evidence has been identified. There is an entire process to be followed after the evidence has been identified to present the evidence in court. This chapter will assist in placing the rest of our study in context of the work that has been done in digital forensics. The chapter has also discussed technology-specific digital forensic fields, such as file system forensics, network forensics, internet forensics, mobile device forensics that will help us determine if database forensics could be solved in a similar way.

Now that the reader has some background information on the digital forensics field the database technology will be discussed. Ultimately, the background of digital forensics and the background of databases will be combined to have an in-depth discussion of database forensics later in this study.

CHAPTER 3 DATABASES

The database chapter is intended to help the reader understand the database technology before discussing the complex field of database forensics. The details about the database technology and relational databases in particular will be useful in later chapters.

3.1 INTRODUCTION

Modern businesses are forced to manage their information in order to survive in the digital era [60]. Organisations without a proper database system are rare. Organisations increasingly rely on information systems (ISs), which results in an increase in the use of databases. A database is a collection of data arranged in a particular way. A database management system (DBMS) is used to organise the data and enables a means for a user of the DBMS to retrieve and access data of the database [10]. Database management systems enable users to access and store data without worrying about the internal representation of databases. Therefore, data are the heart of a DBMS, but data seldom convey useful information on its own. Data have to be interpreted to obtain logical information [11]. The DBMS plays a key role in processing raw data into useful information.

3.2 A BRIEF HISTORY OF DATABASES

Before databases were developed, all data were stored in linear files [25]. Generally, files were created for an application and only utilised by that one application. No information was shared among files and the same data was redundantly stored multiple times in multiple files. Due to significant data demands of organisations, databases developed from file-based systems into the powerful database systems of today.

The US President Kennedy initiated the Apollo moon landing project in the early 1960s. File-based systems were unable to handle the large volume of data required by the project. The North American Aviation (NAA) developed software known as the Generalized Update Access Method (GAUM) to meet the voluminous data processing demands of the project [26]. IBM and NAA developed GAUM into the first commercially available hierarchical model database. This database was named the Information Management System (IMS) and released in 1966 [28]. The hierarchical model structured data as an upside-down tree with the root at the top and leaves at the bottom.

The first general purpose database was developed by ACM Turing Award winner Charles Bachman at General Electric and called Integrated Data Store (IDS). IDS laid the foundation for the network data model. The first specification of the network data model was presented at CODASYL in 1969 [27]. The data model was a slight generalisation of the hierarchical structure where the data were structured as a directed graph without circuits. This allowed the network data model to represent real-world data structures more easily. The network and hierarchical data models represented the first-generation DBMSs and were developed during 1970s, largely to cope with increasingly complex data structures [26].

Dr. Edgar Codd published his paper, “A Relation Model of Data for Large Shared Data Banks” in *Communications of the ACM Journal* in June 1970. Edgar, a graduate in both Mathematics and Chemistry, applied the 19th century discipline of relational calculus to non-procedural expressions of data structures [30]. At that time, software developers brought hierarchical and network DBMSs on the market and although the benefits of the relational data model were recognised by theorists, it took some time before the relational system appeared [29].

The first noteworthy project that put Codd’s theories into practice was the System R project, begun at IBM’s San Jose Laboratory in 1973. The research of the System R project was in the public domain. Larry Ellison saw an opportunity and convinced his boss (Bob Miner) and co-worker (Ed Oates) to start a company, Oracle Corporation, with him. The first commercial relational database, named Oracle, was released by them in 1979. IBM released their flagship relational product DB2 in 1983 [30].

Relational databases were easy to use, and ultimately replaced hierarchical and network databases. The Postgres DBMS was developed by students of the University of California (Berkeley) under the supervision of Professor Michael Stonebraker. The implementation of Postgres began in 1986 as a commercial product and was quite successful. The Postgres project developed into the open-source PostgreSQL DBMS available today [31].

Sybase had a successful DBMS deployed on Unix servers and entered into an agreement with Microsoft to develop a Windows-based version of Sybase. The relationship between the companies turned sour for reasons not publicly known, and both companies walked away with the work developed up until that point. In 1988 Microsoft released the Windows version and called it Microsoft SQL Server [32]. Sybase released Sybase System 10 soon after. Both systems were so similar that Microsoft SQL Server instructors made use of Sybase manuals in class instead of the formal Microsoft SQL Server documentation [28].

Later, the inability of the relational data model to meet the challenges of new applications forced the relational data model to include object oriented features. Hence, in the 1990s the Object Relational Database Management System (ORDBMS) was born. Informix was the first company to release an ORDBMS, and was soon followed by Oracle and DB2 [33]. The next generation ORDBMS was built to support both relational and object oriented features in an integrated way [26].

3.3 DBMS ARCHITECTURE

In the previous section the history of the database was explained. The architecture of the database evolved through recent decades to the database architectures implemented today. This section will discuss the architecture of the DBMS which includes the ANSI SPARC database model, the various components of the DBMS and the execution steps of the DBMS.

3.3.1 The ANSI SPARC Model

Until the late 1960s data were presented to users in its storage format. The database architecture was one dimensional; there was no divide between a presentation layer and the actual storage file [34]. For each new user presentation, a new file was required. This method quickly became inadequate for complex systems.

In the early 1970s data presentation and data storage became more independent. The database architecture was extended to a two dimensional architecture with an internal

dimension for data storage and an external dimension for data presentation. It became possible to provide different presentations based on different sequences. However, as quality of data became a fundamental objective, it became clear that data had to be considered independent of its involvement in application programs.

In February 1975 the American National Standards Institute's (ANSI) Standards Planning and Requirements Committee (SPARC) published an interim report explaining the architecture they had developed for information systems [35]. They introduced the ANSI/SPARC database architecture which was a three dimensional architecture. Along with the internal dimension and external dimension, a new conceptual dimension was introduced. In a practical context, the various levels of the ANSI/SPARC architecture can be represented as illustrated in Figure 3.1. The various dimensions (or schemas) are explained as follows:

3.3.1.1 External Dimension

The external dimension represents a user's view of the database. There can be various user views of the single conceptual dimension, as displayed in Figure 3.2. The external level can be modified according to the needs of an individual user [36].

3.3.1.2 Internal Dimension

The machine view of the database is represented by the internal dimension. The internal dimension contains the description of how the data are stored. This includes record implementation, storage techniques and field implementation techniques [35].

3.3.1.3 Conceptual Dimension

The conceptual dimension is the database designer's view of the database and completely unrestricted. The database designer is able to see all structures within the database. This dimension was added by the ANSI/X3/SPARC study group as the central component of the architecture [37]. All data structures are defined at this level, including data definitions, table structures and object classes suitable for the database management system used.

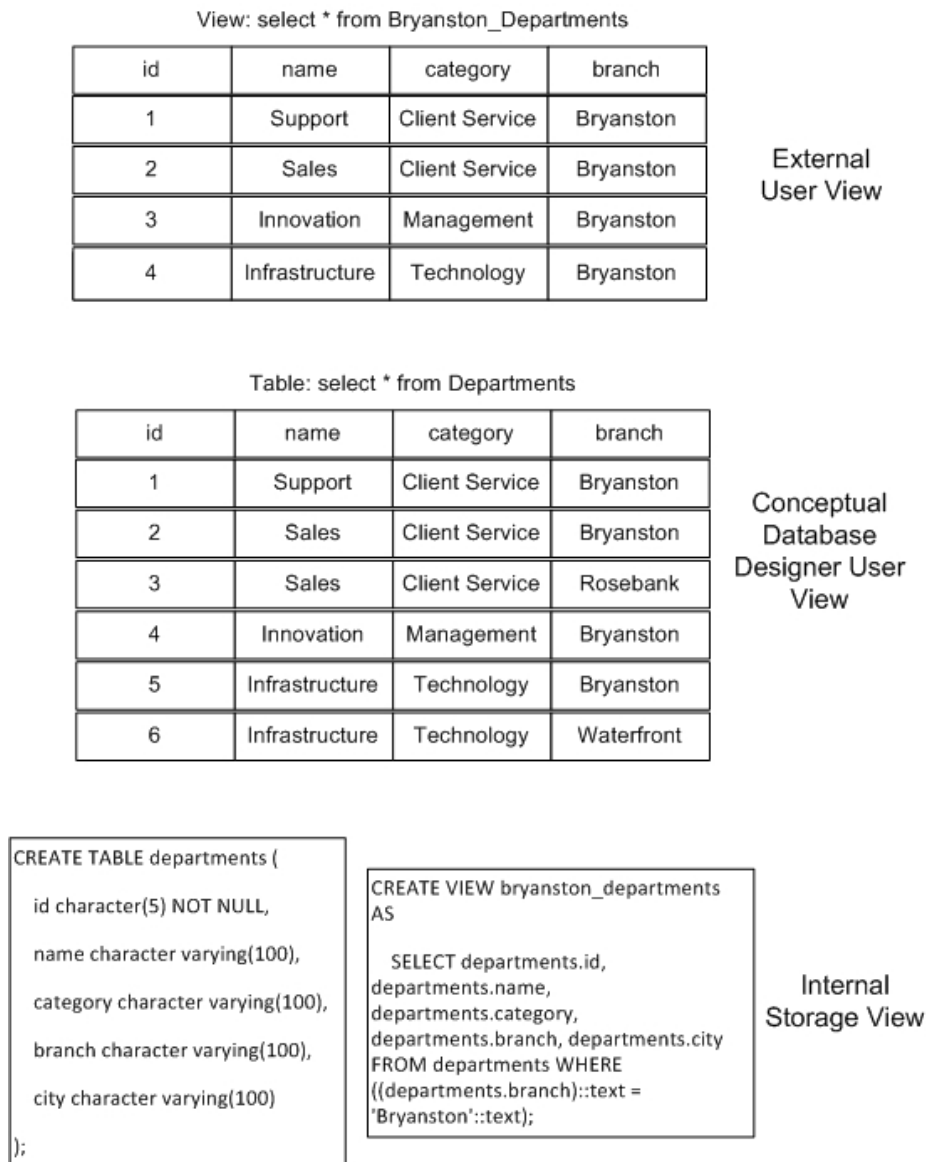


Figure 3.1. A practical illustration of the ANSI/SPARC database architecture

3.3.1.4 Mapping

Figure 3.3 displays the ANSI/SPARC architecture as a three dimensional graph. Each dimension has a mapping onto the next dimension. The Conceptual-Internal (C-I) mapping defines the correspondence between the conceptual view and stored data. It enables the DBMS to find the actual stored record that constitutes the logical record in the conceptual schema. The External-Conceptual (C-E) defines the correspondence between the user view

and the conceptual view. Any number of external views can be mapped to the relevant part of the conceptual schema. As Figure 3.2 above illustrates, there could be one C-I mapping and multiple C-E mappings. The Internal-External (I-E) mapping is an abstract mapping which has to do with what tasks are performed to query data.

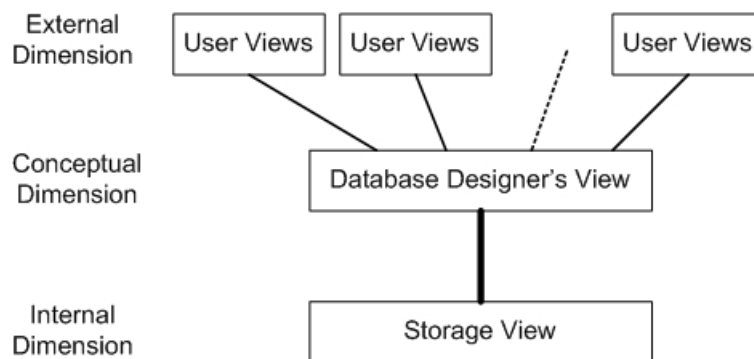


Figure 3.2. A simplified graph of the ANSI/SPARC database architecture

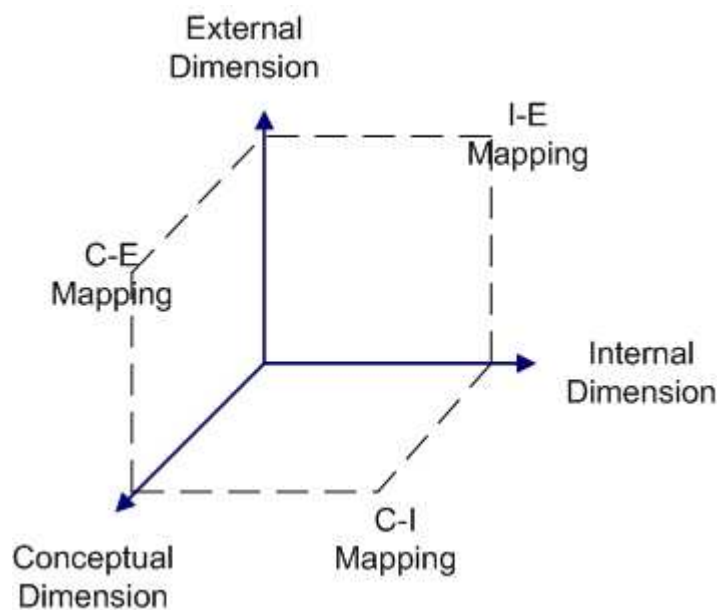


Figure 3.3. The ANSI/SPARC database architecture as seen as a three dimensional graph

3.3.1.5 Advantages of the Three Dimensional Architecture

Why is the ANSI/SPARC important in database architectures? The ANSI/SPARC was one of the early progressions made towards data independence from user applications [34].

The ANSI/SPARC architecture brought about the following advantages [26]:

- Each user can change the way they view the data from the database without influencing other users from the same database.
- The user's interaction with the database is independent from the physical storage organisation of the database.
- A database administrator (DBA) is able to change the physical storage organisation of the database without influencing the users. This phenomenon is called physical data independence.
- The conceptual structure, or the database design, can be changed without influencing the users. This phenomenon is called logical data independence.

3.3.2 Components of the DBMS

Now that a general architecture of the DBMS has been discussed, we turn our attention to the various components in the DBMS architecture in order to understand what functionalities make the DBMS a useful tool. The DBMS accepts commands from various user interfaces, constructs a query evaluation plan, execute plans against the database and returns a result [26].

The software modules that the DBMS uses are described as follows:

- Query Processor. When a user query is received, the query processor checks if the query is valid and if all queried resources are available in the database. If the query is valid, the query processor generates an execution plan. Any number of execution plans can be generated depending on the number of operations in the query and the number of methods that can be used to evaluate each operation. The execution plan with the minimum cost is selected by the DBMS by making use of query optimisation [38]. The query processor translates the user query into an optimal low-level instruction set.

- **Run Time Database Manager.** User queries received via the query processor are converted from the user's logical view to the physical file system. This is done via various sub-systems within the run time database manager. The authorisation controller checks if the user has the required access to process the operation. The integrity checker checks for necessary integrity constraints for all requested operations. The transaction manager requests and releases locks according to a locking protocol, and schedules the execution of transactions. The scheduler ensures that concurrent operations do not conflict with one another. Lastly, the data manager is responsible for handling data in the database which includes recovery mechanisms in case of failures and buffering to transfer data between main memory and secondary storage.
- **DDL Processor.** The Data Definition Language (DDL) is responsible for handling the *CREATE*, *ALTER* and *DROP* SQL statements [39]. The DDL processor converts the DDL statements into a set of tables with metadata.
- **DML Processor.** The Data Manipulation Language has three SQL statements: *INSERT*, *UPDATE* and *DELETE*. The DML processor converts the DML statements into object code for database access.

3.3.3 Execution Steps of the DBMS

Now that the architecture and components of the DBMS have been reviewed, the execution steps of a query will be discussed.

The following query compilation steps are executed when the user requests to access the DBMS [40]:

1. The user issues a query to the DBMS.
2. The syntax-checking phase commences, the system parses the query and checks that it obeys syntax rules. Furthermore, the objects in the query are matched with the views, tables and other structures in the database to ensure they exist. Finally, the privileges of the user are checked to ensure that the query may be executed.

3. The query optimisation phase begins when existing statistics of tables and indexes are referenced to work out the optimal execution plan. An optimal access plan is selected to perform the query.
4. Finally, the access plan is put to effect in the execution phase and the physical storage is accessed to retrieve the answer to the query.

3.4 THE RELATIONAL DATA MODEL

The previous section described the structure of database systems in general. In this section we will focus on the relational data model's structure and concepts. We focus on the relational model because the relational data model is the foundation (or large parts of the foundation) of the database field in particular [1]. Furthermore, the relational model is the most prevalent of all database models in modern business together with well-established ANSI and OSI standards. This section will start off by introducing the mathematical concept of the relational data model, as developed by Codd. The structure of the relational data model will also be discussed and finally the structured query language is explained.

3.4.1 Relational Algebra

As mentioned before, the relational data model was developed by Codd by making use of relational algebra to query relations. Database relations are queried by the relational data model by making use of primitive operators from relational calculus [45]. In relational algebra, operators manipulate relations to form new relations [41]. In the relational data model, a relation is a table or a result set that looks similar to a table. For example, the union operation combines the tuples of one relation with the tuples of a second relation to produce a third relation. Tuples can be equated to rows of a table in the relational data model. Originally, eight operations were proposed by Dr. Codd, but several others have been developed since [26]. Some of the fundamental relational algebra operators are discussed here.

3.4.1.1 Union

The union of two relations is formed by adding the tuples of one relation to the tuples of a second relation to produce a third relation or result. The union of relations A and B is

denoted as $A + B$. The union operation creates a new table by placing all rows from two source tables into a result table [42]. Relations must be union compatible, meaning each relation should have the same number of attributes (columns) and the attributes should be of the same domain or type.

3.4.1.2 Projection

Projection is an operation that selects specified attributes (columns) from a relation. In relational algebra terms, the selection operator π is used to obtain the desired projection, according to Codd [43]. Thus, if R is a n -ary relation (table with n columns) and L is a list of k attributes (columns to project), then $\pi_L(R)$ is the k -ary result relation. A query language equivalent of the projection operation is: *SELECT attribute1, attribute2 FROM table*.

3.4.1.3 Selection

As the projection operator takes a vertical subset of the relation, the selection operator takes a horizontal subset of the relation. The selection operator identifies the tuples to be included in the result subset. The selection σ applied over a relation R with respect to formula F is defined as $\sigma_F(R)$. The formula F is constructed by Boolean logical connectives *NOT*, *AND* and *OR* equivalent to algebra operators: \cup , \cap , and $-$ [44]. A selection query is written as: *SELECT target data FROM table WHERE formula*.

3.4.1.4 Join

The join operator is a combination of the product, selection and projection operations. The joining of two relations A and B is calculated by first forming the product of A times B ; then the selection is applied to eliminate some tuples and finally, the projection removes some attributes from the result relation. A natural join can be expressed as $A * B$, but projections and selections can be applied to the natural join to eliminate tuples or attributes: $\pi_L(A * B)$. A join query is written as: *SELECT attribute1, attribute2 FROM table1 LEFT JOIN table 2 ON point of ambiguity WHERE formula*.

3.4.2 Relational Data Model Structure

Based on the relational algebra discussed in the previous section, the relational data model was developed into more than one hundred commercial database management systems since the invention of the model. The relational model has important structure concepts that will be discussed in this section in order to understand the fundamentals of the relational model of today.

3.4.2.1 Relation Concept

A relation is a two dimensional table. Each column in the table contains data regarding an attribute. Each row or tuple in the table contains a collection of data of various attributes. The terms *relation*, *tuple* and *attribute* stem from relational mathematics, as mentioned before. Users commonly refer to these terms as *table*, *row* and *column* respectively. For a table to be a relation it must meet certain restrictions [41]:

- Firstly, the cells of the table should be single-valued, meaning groups or arrays are not allowed as values.
- All entries in a table must be of the same kind.
- Each column has a unique name and the order of the columns is insignificant.
- Finally, no two rows in a table may be identical.

3.4.2.2 Functional Dependencies

A functional dependency is a relationship amongst columns of tables. Given the value of attribute X, we should be able to obtain the value of attribute Y, when attribute Y is functionally dependent on attribute X [46]. This definition is similar to the definition of a function in mathematics. Figure 3.4 compares a function where Y is functionally dependent on X with a function where Y is not functionally dependent on X. From the figure it is clear that, for each value of X, there is a value of Y; thus, Y is functionally dependent on X. In a similar way, data can be queried in the database due to functional dependence amongst attributes. Identification numbers are often used in tables to find functionally dependent data, such as student courses linked to a student ID.

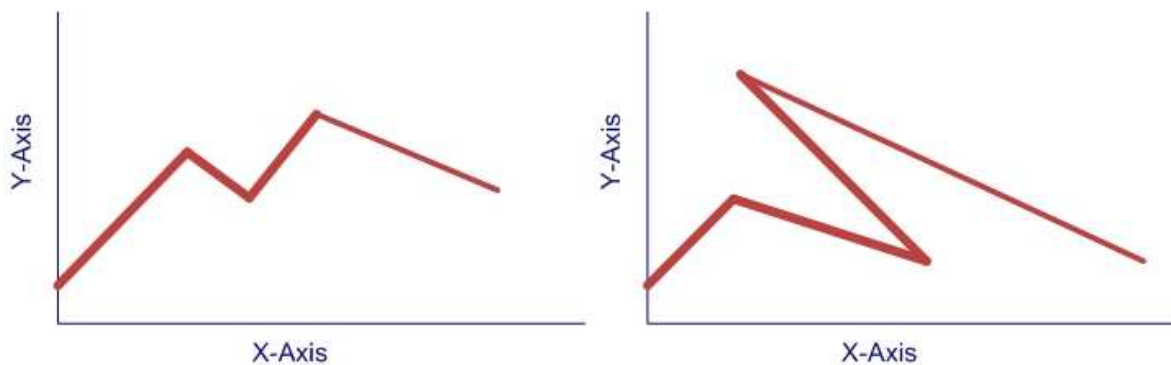


Figure 3.4. A graphical representation of a functional dependent function (left) and a function that is not functionally dependent (right)

3.4.2.3 Keys

A relation always has a unique identifier or group of unique identifiers whose values are unique for each tuple in the relation. Each tuple is unique and can be identified by a key consisting of one or more attributes. There are several types of keys in the relational database structure, namely:

- Relational key. This key is a set of one or more columns that are unique in the table. The set of keys is required to be unique, non-redundant and may not be null.
- Candidate key. Candidate keys are present when more than one or more than one group of columns can be defined as the relational key.
- Primary key. This variation of a candidate key is used to uniquely identify a row, but other columns may still uniquely identify the row as well.
- Foreign key. A foreign key is a column or set of columns in one table that matches the candidate key of another table.

3.4.3 Structured Query Language

A language is required to extract data from the relational database and manipulate data in the relational database. Structured Query Language (better known as SQL) is a relational database management system (RDBMS) -specific programming language designed to define relational schemas and tables as well as provide data manipulation capabilities [48]. SQL does not exist outside the RDBMS and cannot be used as a standalone programming language.

The first ANSI SQL standard was published by the National Standards Institute in 1986 to bring greater conformity amongst vendors of RDBMSs. A variety of SQL behaviours and syntax structures across many products was formalised by SQL. SQL became even more important as the popularity of open source RDBMSs grew. SQL is an open language that is not owned or controlled by a single company [49]. Different dialects of SQL do exist (PL/SQL, Transact-SQL, MySQL etc.) amongst various vendors of RDBMSs caused by vendor customer-specific requirements and capabilities developed before SQL was standardised, but the concept of SQL remains very much the same across vendors.

SQL can be divided into four components that are used to extract or manipulate data. The discussion of how each statement is compiled is beyond the scope of this chapter and more information can be found in [48].

- The Data Query Language (DQL) extracts data from the RDBMS by making use of the *SELECT* statement.
- The Data Definition Language (DDL) is used to create, modify or destroy tables within the RDBMS by making use of the *CREATE*, *ALTER* and *DROP* statements respectively.
- The Data Manipulation Language (DML) is used to manipulate rows by using the *INSERT*, *UPDATE* and *DELETE* statements.
- The Transaction Control Language (TCL) does transactional tasks by using the *COMMIT*, *ROLLBACK* and *SAVEPOINT* statements.

The structured query language can be used in two ways [50]. The first method is the *interactive* way where a user types in a SQL statement on the spot and the database processes the query immediately. The interactive way is used by programmers and users who want to create data reports themselves. An interactive query can be requested either on a terminal interface (like MySQL) or on a modern graphical interface (like WinSQL).

The second way to use SQL is the preprogrammed way where SQL statements are embedded in an application that is written in another programming language. Three

components are important in preprogrammed processing of SQL statements: the users, the application and the database server. The database server is responsible for storing and accessing data on the disk. The application requests data from the database server and the database server delivers the result. The result of the SQL statement is displayed to the user in a way defined by the application.

3.5 DATABASE DESIGN

3.5.1 Introduction to Physical Database Design

As data grow dramatically due to the expansion and ubiquitous use of networked computers in homes, businesses and the industrial world, good database design becomes more vital. The difference between bad database design and good database design can be up to 50 times when running queries [51]. The need for database design stems from large volumes of data. Best practice design, like the use of indexes, will have no visible effect on a table with 20 rows, but as the volume in the database rises, the database design becomes critical in order for the database to deliver results in time. The database life cycle incorporates the basic steps involved in designing a global schema of the logical database.

The database life cycle consists of the following phases [52]:

- Requirement analysis. This phase necessitates the designer to produce a requirement specification with a list of database requirements consisting of data required for processing, data relationships, and the software platform for database implementation.
- Logical design. A conceptual data model diagram that shows all the data and their relationship (called a global schema) is developed with techniques such as ER or UML. This design is ultimately transformed into normalized tables.
- Physical design. This phase involves the selection of indexes, partitioning, and clustering of data. Denormalisation is also done in this phase to make frequently required queries more efficient.

- Database implementation, monitoring and modification. After the design is completed, the database schema is created by making use of the data definition language. The data manipulation language is used to query and update the database.

In the following sections database design techniques that are used during the database life cycle to make the database more efficient will be discussed.

3.5.2 Indexing

A library has a catalogue of all books available. If you want to find a book, you can quickly search through the catalogue and find the location of the book instead of searching through every single isle in the library. Database indexes work in a similar way to find data. The drawbacks of creating database indexes is the additional space required to store the index and the additional time it takes to update a table, but the speed gained when querying data overshadows these drawbacks by miles [53]. An index should be created if the benefit is greater than the cost. The B+tree index has evolved to become the most popular indexing method for relational databases. Various other indexes exist and more information can be found in [51]. As a general rule of thumb, the primary key and foreign key of tables should at least be indexed. Furthermore, a column of a table should be indexed if the column frequently appears in the *WHERE* clause of queries. However, it is important to investigate whether the index is improving performance by running new queries on the relevant table.

3.5.3 Query Optimisation

Query optimisation should occur on two levels which are by the database designer and by the DBMS. The DBMS optimises queries as a best effort service, but if queries contain complex conditions, the database designer should minimise query execution time with good database schema and query design. The DBMS query optimiser is a cost-based optimiser which analyses a number of candidate execution plans, estimates the cost of each of the plans and selects the plan with the lowest cost [54]. The query optimiser has the biggest effect on the performance of a database, because the selection of the right query

can be the difference of a query executed in milliseconds and one of minutes or even hours.

The SQL statement is parsed by checking the SQL syntax and translates the query into an initial tree which contains the high level steps to perform the query. The binding phase is concerned with name resolution to ensure all objects in the query exist. In order to avoid rework, the cache of the DBMS is checked to ensure that an execution plan of the query is not stored before the SQL statement is optimised [55]. If no plan is stored in cache, the query is transformed into an internal representation that is more suitable for processing. This internal representation is called an algebra tree [26]. The query optimiser now needs to select a plan to execute on the algebra tree. Given that the query optimiser cannot evaluate each possible plan, it has to find a balance between the optimisation time and the quality of the plan.

A query cost optimisation algorithm is used to find a quality plan. Several artificial intelligent search algorithms or query cost optimisation algorithms exist to find the minimum cost execution plan, such as the Distributed INGRES algorithm, Distributed R* algorithm and the SDD-1 algorithm [56]. Logical operations such as *JOINS* can be translated into several physical operations such as *NESTED LOOPS JOIN* or a *MERGE JOIN*. It is from these physical operations which the query optimiser must find the minimum cost execution plan. The query optimiser delivers a tree consisting of a number of physical operators as output which contains algorithms to be performed by the database execution engine.

To make efficient use of the DBMS query optimiser, the database designer has to design SQL statements to make full use of query optimisation. Some SQL variations are easier to optimise than others although they deliver the same result set. During the course of this study we have determined that one complex query will perform much better than a lot of simple queries.

Furthermore, two incorrect SQL statements are displayed in Figure 3.5 to show a join predicate on expressions that will not make full use of the query optimiser because the

processing needs to be done before the equal sign. Some of the best practice techniques to design SQL queries are listed in [57].

```
WHERE SALES.PRICE * SALES.DISCOUNT = TRANS.FINAL_PRICE  
WHERE UPPER(CUST.LASTNAME) = TRANSNAME
```

Figure 3.5. Two SQL statements *WHERE* clauses that will cause the query to not make full use of the query optimiser [57]

3.5.4 Decomposition

A functional decomposition is the process of breaking the functions of an organisation into greater levels of detail. With decomposition, one function is described in greater detail by a set of supporting functions [26]. Decomposition in databases is done to avoid redundancy by dividing modules into their smallest form. The decomposition of relation R is achieved by dividing relation R into two or more relations, each one containing a subset of the attributes of the relation R and together they include all the attributes in R .

3.5.5 Normalisation

One of the principal objectives of databases is to ensure that each data item only appears once within the database [58]. The reasons for this are (1) the amount of storage space for data should be minimised and (2) to simplify maintenance of the data. For example, if data existed in more than one place, there would be a possibility that they might become different in time due to incorrect maintenance. The normalisation process can be utilised to organise data into a standard format to avoid processing difficulties. Database normalisation is essentially a set of rules that allows a related and flexible database design [59]. The sets of rules in normalisation are called ‘normal forms’. If the first set of rules is adhered to, the database is in the first normal form and if the third set of rule is adhered to, the database is said to be in the third normal form.

The main rules of the first normal form are:

- Eliminate repeating information.

- Create separate tables for multi-valued attributes.

The main rule of the second normal form is:

- If the primary key has more than one field, ensure that all fields are dependent on the whole key.

The main rule of the third normal form is:

- Remove all fields that depend on non-key fields.

Further normal forms include the Boyce-Codd Normal Form, Fourth Normal Form, and Domain-Key Normal Form. These forms can be found in [58].

3.6 DATABASE SECURITY

3.6.1 Background

Although information security is well-researched, computers and other electronic devices are used increasingly to commit crimes against persons, organisations or even property [61]. If money could be imagined to exist on a computer network, it would exist in the database server [8]. Almost all important information in our lives like our bank accounts, medical records, employment records, tax records and pension fund information is stored on a relational database management system [62]. Because data stored on databases are of such critical value, there is more than enough reason for individuals to copy, view or change the data on a database.

Database security is considered to be one of the most vital information security issues to be managed because our most sensitive data are stored on databases [64]. A breach of security in databases can be devastating for both the individual and the wider society. The recent increase in Web-based applications and information systems has further augmented the risk of a database exposure in such a way that data protection is more important than ever [63]. This section will discuss various key database security components used in DBMSs, such as access control, data encryption and statistical database security.

3.6.2 Threats

All parts of the database system, including the database, the hardware, the operating system, the network, the users and even the building should be secured against threats [26]. Various threats exist that should be addressed in a comprehensive database security plan [65]. The first threat is loss of availability where the database system cannot be accessed by users. Loss of data integrity causes corrupted data which could be catastrophic to an organisation because of wrong decisions made with wrong data. Loss of confidentiality or secrecy refers to when, for example, an organisation's secret data is exposed, and the strategic plans and strategic data are revealed. Loss of privacy refers to when the data in a database are exposed due to a lack of protection against users. Loss of privacy may lead to blackmail, public embarrassment and stealing passwords. Theft and fraud are threats to an organisation or owner of the database. Accidental losses are other threats to the database and refer to unintentional harm caused to the database due to human error or software and hardware-caused breaches. The following sections describe methods to limit the risk of a database security breach.

3.6.3 Access Control

A key component to ensure data protection in databases is the access control mechanism. This mechanism is used to check whether a subject has the authorisation to perform actions on the data [66]. The typical database access control mechanism is more complex than the access control mechanisms of Windows [67]. For example, Oracle 10g has 173 system privileges of which six can be used to take complete control of the database system. Two database access control models are discussed here. They include the discretionary access control model and the mandatory access control model.

3.6.3.1 Discretionary Access Control Model

The discretionary access control model gives the owner of the database the privilege to grant or revoke authority to objects in the database. Access rights such as *SELECT*, *INSERT*, *DELETE*, *UPDATE* or a combination of these can be granted to users on specified objects in the DBMS [26]. The advantage of the discretionary access control

model is that it is very flexible because any set of customised rights can be given to any user and users can pass their privileges on to other users.

A disadvantage of the discretionary model is that right revocation might not have the planned effect. If subject S_1 grants privileges to S_2 on object O_1 , S_2 may grant the same privileges to S_3 . S_1 may then assign the same privileges to S_3 . If S_2 later changes her mind and revokes the privileges of S_3 , the privileges will not be revoked because S_3 still has the same privileges from S_1 to access O_1 [68].

3.6.3.2 Mandatory Access Control Model

Both objects and subjects have a classification level under the content access control model [69]. Whenever a subject tries to access an object, an access rule from the operating system decides if access should be granted to the user. Unlike the discretionary access control model, access rights cannot be passed on from user to user. This enables the mandatory access control model to implement enterprise-wide security policies on the database [68]. What is gained in enterprise-wide security is lost in flexibility to grant certain user access to certain data. Implementation of the mandatory access control model has been named ‘multilevel relational model’. A multilevel relational model is characterised by different tuples in a relational having different access classes [66]. The model is further complicated if different attributes in a tuple can have different access classes. Typically, a subject who has access to class c in the database may view all rows with a c access level or lower.

3.6.4 Data Encryption

Another method of minimising threats against the database is by data encryption. Data encryption may occur at different levels, such as data encryption within tables, data encryption at rest (data encrypted in stored files in the operating system) and data encryption on the wire (data encrypted when communicating over a network) [70].

As a general rule we can assume that the stronger the data encryption technique and the more data that are encrypted, the more CPU power will be required to encrypt and decrypt

the data of the database. A database administrator can either decide to encrypt data or hash data. Common encryption functions used in databases include (from weakest to strongest) DES, TRIPLE_DES, RC4, DESX and AES256. Modern DBMSs like SQL Server and Oracle 11g make use of transparent data encryption (TDE) where the whole DBMS is secured by a protected database encryption key (DEK). TDE performs all cryptographic functions at the I/O-level within the database and relieves developers of creating custom codes to encrypt or decrypt database data [71].

3.6.5 Statistical Database Security

Lastly, statistical databases secure data about individuals by only allowing access to aggregation functions in queries and hence only deliver statistical results as output [26]. Only aggregation functions like SUM, AVG, COUNT, MAX and MIN can be used to select data in a query. There are ways to attack a statistical database and can be found in [10].

3.7 DATABASE TRENDS

It seems that relational database management systems will remain the most popular DBMSs for the near future due to the amount of expertise available for RDBMSs, the maturity of the product and the support that exists for RDBMSs [72]. The RDBMS has successfully evolved the RDBMS to adapt to new technologies, such as object orientation. A technology that could have posed a threat to RDBMSs has been integrated into the RDBMS to become a DBMS feature instead of a new DBMS technology. Another DBMS system that is receiving a lot of attention these days is NoSQL DBMSs [73].

These DBMSs break the traditional rules of the RDBMS. They do not expect data to be normalised, but instead the data are accessed by a single application and live in one large table [74]. These systems are designed to manage terabytes of data due to the huge data demands of modern systems. Scalability is a key feature of NoSQL DBMSs and is achieved by MapReduce – a parallel programming model that allows distributed processing on large data sets by a cluster of computers. Only time will tell what the DBMS of the future will be.

3.8 CONCLUSION

This chapter provided an overview of databases with specific focus on the relational database. This study will ultimately design an approach to execute a forensic investigation on a relational database and therefore it is important to have an understanding of the components and inner-workings of the relational database. Chapter 2 discussed digital forensics and this chapter discussed databases, while the next chapter will combine the two fields to discuss database forensics.

CHAPTER 4 DATABASE FORENSICS

4.1 INTRODUCTION

Computers and other electronic devices are used increasingly to commit crimes against individuals, organisations or even property [61]. In many cases the proof can be found against a suspect of digital crime by conducting a digital forensic investigation. In recent years digital forensics has grown from a relatively ambiguous tradecraft to an essential part of many investigations [75]. Where a breach of security or unauthorised use of a system has occurred or corruption has taken place on a digital system, digital forensics is required to analyse the digital system and consequently determine what has happened on the system in a way that could be presented in court. Digital forensic tools are now used by examiners and analysts on a daily basis. Development in digital forensic research, tools and processes have been very successful over the last decade [75]. It is therefore odd that so little literature exists on database forensics.

Modern businesses are compelled to manage their information in order to survive in the digital era [60]. Organisations without a proper database system are rare. Organisations rely heavily on information systems (IS) which results in an increase in the use of databases. Databases have become an essential part of all industries today [11]. Along with digital advances, the use of databases have grown extremely fast. Databases usually hold critical data [8,9,10,11]. There is more than enough reason for malicious individuals to copy, view or change the data on a database because data stored on databases are of critical value.

We argue that the field of database forensics is very important due to the following motivations:

- Database systems are used extensively throughout the business world on a daily basis and contain critical data.
- There is a tendency to store information about crime on digital devices, commit crimes on digital devices or perform actions on digital devices; this requires investigations to find out what has happened [23]. This behaviour has warranted the development of the field of digital forensics.

- The field of incident response on databases has grown over recent years due to the influx of requests to recover a database or determine what has happened on a database.

In a timeframe where the fields of databases and digital forensics have enjoyed a lot of attention, the field of database forensics still lags behind significantly [76]. There is a lack of tools available to do effective database forensics. Research was done to highlight the lack of work done in the field of database forensics [23]. The limited literature available about database forensic is case study-driven rather than based on a generalised theoretical foundation. This means that some literature discusses case studies of specific database attacks and how to react to such attacks, but virtually no literature exists on how to approach a database forensics crime scene in an ordered and forensically sound manner. This lack of literature might be due to the inherent complexity of databases that are not understood in a forensic context as of yet [23]. Database forensics has been labelled as a new demand on digital forensics [11].

Previous chapters have stated that extensive research for databases as well as digital forensics exists. They also state that the database is an invaluable part of our technology dependence in modern times. Given this information, it comes as a surprise that there is such limited information about database forensics.

The remainder of this chapter will attempt to contribute research to the void that exists in the field of database forensics. The next section will discuss the dimensions of database forensics. The dimensions of database forensics will familiarise the reader with the range of database forensic types and place this study in context of work already completed in the field of database forensics. Thereafter database forensic tools that exist today will be discussed. Database forensic tools are discussed here based on the assistance they might provide to a forensic examiner during an investigation. Thereafter the database forensic process will be discussed. A forensic process assists a forensic investigator in planning and completing investigations in an ordered fashion. Therefore, a database forensic process deserves our attention. Finally, we will turn our attention to database forensics literature

which will assist in solving the problem of this particular study. This section will also initiate a discussion towards how database forensics could be approached for compromised DBMSs.

4.2 DIMENSIONS OF DATABASE FORENSICS

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 [77]. In database forensics it is important to be able to retrace operations performed, reconstruct deleted information or reconstruct compromised data on the database.

The question that comes to mind is what type of databases and what type of attacks are investigated on these databases? Unfortunately the question cannot be answered by a couple of forensic examples alone. It has to be assumed that the options are numerous and an effort to jot down all possible scenarios of database forensics will consume a large amount of time through research and testing of various DBMSs. Such a list of possible forensic scenarios will become outdated daily, as database systems are developed on a daily basis.

A better plan would be to rather work on an *approach* that might cater for all possible database forensic environments in a wide variety of DBMSs. Even though a list of possible database forensic scenarios might be a flawed notion, there is potential to divide databases into different categories that may warrant a forensic investigation. These categories will assist in comprehending the problem space of database forensics.

The discussion of the dimensions of database forensics is useful to the reader for a couple of reasons. The first reason to talk about the dimensions of database forensics is to place this study in context of work that has been done in the field of database forensics. By doing this the reader can obtain an improved view of where this study fits into the database forensics field. Another reason to discuss the dimensions of database forensics is to describe just how multi-faceted the field of database forensics is. This will prepare the reader to view this study in the context of such a divergent field.

In a recent study the field of database forensics was divided into three dimensions [78]. According to the study the dimensions of database forensics include compromised databases, damaged databases and modified databases. The study focused on the current research that exists in database forensics.

The division of database forensics into three dimensions may potentially assist a forensic investigator in approaching a database forensic investigation more effectively when it can be determined beforehand to which dimension the database at hand belongs. This is done because the field of database forensics is extensive and the problem could be made more manageable by linking the database to be investigated to a type of database forensic dimension.

The disadvantage of this approach is that new research in the field of database forensics may arise that may introduce a new dimension of database forensics. However, the field of database forensics is in its infancy and a solution needs to be found for the current lack of work on database forensics.

The following sections will discuss the various dimensions of database forensics that are available today.

4.2.1 Damaged or Destroyed Databases

Damaged databases represent DBMSs that have been damaged by deleting, editing or moving the data contained within the DBMSs or data files of the DBMSs [78]. Damaged or destroyed databases often require experts to extract evidence from the databases, because normal database users usually do not have the skills to make sense of damaged or destroyed databases. A database could be damaged by uninstalling the DBMS, deleting files from the DBMS installation, running commands in the DBMS that corrupts data etc.

As opposed to the other dimensions of database forensics, there is some literature that relates to database forensics in the damaged database dimension. Most of the research on database forensics falls into this category. It is important to note that investigations based

on database forensic have been conducted in practice. Perhaps the most complete practical guide that relates to database forensic investigation was compiled in a series of papers by David Litchfield [79,80,81,82,83,84,85]. In this series of papers practical steps are discussed to extract evidence from an Oracle DBMS. This information is very specific towards the Oracle database management system and interlinked with database security.

In another study, technical methods were proposed to identify when data on an Oracle database have been modified. Methods were also proposed to recognise vulnerabilities in the Oracle database [87]. Incident responses that relate to database forensics in some way were conducted in corporate environments where clients had asked professionals to investigate their databases for possible attacks [88].

In other literature, data mining techniques have been proposed to simplify data recovery on large datasets [89]. These large datasets may relate to databases in some way but data mining techniques that apply to digital investigations are in their infancy stage and limited research has been done in this field. Database forensics has been discussed for the InnoDB DBMS [90]. This study focused on InnoDB file formats and how to recover data from the data files of an InnoDB DBMS.

The preceding information about damaged databases confirms that the research done on database forensics is divergent and often only partially focuses on database forensics. The information is also documented separately for some unique DBMS but little or no attempt is made to document database forensics as a structured methodical field.

The current literature on damaged databases also focuses on understanding security risks posed to databases and how to recover from such attacks. Information is available on security flaws in database management systems that can be exploited to receive unauthorised information or launch attacks on databases [8]. The flaws in the security of well-known database management systems like Oracle, DB2, Sybase, MSSQL, etc. are illustrated. This information is useful in order to gain knowledge on the type of attacks that will typically be investigated.

4.2.2 Compromised Databases

The second dimension of database forensics discussed here is compromised databases. Compromised databases represent DBMSs where the metadata or software of the DBMSs have been modified by an attacker. Data are inserted into a structure of code and metadata to store information. The DBMSs even allow users to create some metadata of their own by creating metadata structures such as tables, views, triggers etc. Possibilities exist to alter the metadata of the DBMS in such a way that a false representation of the data is presented to the user.

Presently incident response of databases may query the database without even considering that the data presented by the DBMS may be a false picture of what is truly going on in the DBMS. In the context of forensic evidence this poses a challenge to the forensic investigator. Evidence from the DBMS will ultimately be used to prove that something has occurred in the DBMS but if the evidence cannot be trusted due to compromised metadata, the evidence could never be used in a court of law.

The different types of metadata and data that exist in a DBMS will be discussed later in this chapter. For now it is important to note that this metadata might be compromised to affect evidence in some way. Traditional sources of evidence from the DBMS, such as log files, DBMS dumps, DBMS query results etc., cannot be trusted until a forensic investigator could be assured that the DBMS has not been compromised. Metadata could be edited to alter operators and will be proved in detail in later chapters.

Furthermore, the code of the DBMS could be changed to manipulate the DBMS to deliver any result that a criminal wants it to deliver. Source codes for various open source DBMSs, such as MySQL and PostgreSQL, are available without cost and can be altered in multiple ways. A criminal who changes the source code will most probably execute it in such a way that the changes will go unnoticed by the users of the DBMS.

Research on compromised DBMSs is rare but some literature has been made available in recent years. A forensic investigation on a DBMS has been compared with a forensic

investigation on an operating system [23]. In operating systems, the data allegedly retrieved from the system may not be what is really on the operating system. Similarly, the output from a DBMS is a function of the metadata in the DBMS.

The same study provided two examples of a compromised database scenario. The first example was that of a cracker who breaks into a shop's database, and then interchanges the cost price and selling price columns. This leads to a significant loss for the shop, as products are sold at cost price. In this scenario the data of the DBMS have not been changed, but the metadata were compromised. This example has proven that it is possible to do and is included in a following chapter.

In the other example the criminal attempted to hide evidence in a table of the DBMS in such a way that it would not make sense to the normal users of the database and would probably go unnoticed. The criminal could then create a view to make sense of the hidden data. Removing this view would leave no sensible evidence behind. The metadata created by the criminal thus constructed a view of data that could be useful in a forensic investigation.

Another study that focuses on database incident response encourages the use of a checklist during the performance of a live response on Oracle databases [82]. This checklist (maybe unintentionally) includes various tasks to confirm that the Oracle DBMS has not been compromised. A similar checklist may be a useful approach to determine to which dimension of database forensics a database belongs.

In a presentation at the UK Oracle User Group (UKOUG) it was stated by an Oracle forensics expert that tools that could be used to extract forensic evidence from an Oracle DBMS include database dumps and simple DBMS queries [86]. These tools are very convenient to use and are understood by most database users. However, the present study argues that the notion of a compromised database has not been considered in this instance. These tools of the DBMS cannot be used if it is uncertain whether the DBMS may have been compromised.

4.2.3 Modified Databases

The third dimension of database forensics is modified databases. Modified databases represent a DBMS that has not been compromised or damaged but has undergone changes due to normal business processes since the event of forensic interest has occurred [78]. DBMSs are used all over the world and process a large number of queries or other database tasks daily, depending on the number of requests sent to the DBMS. If an incident of forensic interest occurs on a DBMS, the DBMS will not stop executing database tasks if the DBMS is not explicitly instructed to do so. Therefore, a forensic investigator might have the challenge of extracting evidence from an environment which has changed since the incident of forensic interest has occurred. In a traditional forensic murder scene the forensic investigator should consider any changes that have occurred on the crime scene since the crime has been committed. It is logical to apply the same principle to databases.

Some work has been published on reconstructing a database to an earlier time even though modifications on the database have occurred [91]. Inverse functions of each operator of relational algebra have been defined. The same paper also explains how a relational algebra log could be generated from a complex database log by transforming queries into operations involving relational algebra. Reconstruction of a database is done by making use of inverse operators and a relational algebra log.

4.2.4 Orthogonality of Dimensions in Database Forensics

The three dimensions of database forensics can be envisioned as being orthogonal. A database being investigated may belong to one or more dimensions in this orthogonal structure [78]. The database being investigated may belong to all three dimensions. The orthogonality of the dimensions of database forensics is important because a forensic investigator should not limit a database investigation to a single dimension if the database contains elements of more than one dimension. The investigator will be empowered to understand the problem space and its limits when the database could be placed within one or more dimensions. This means that the forensic challenge at hand becomes easier if the forensic investigator knows the extent of the forensic challenge.

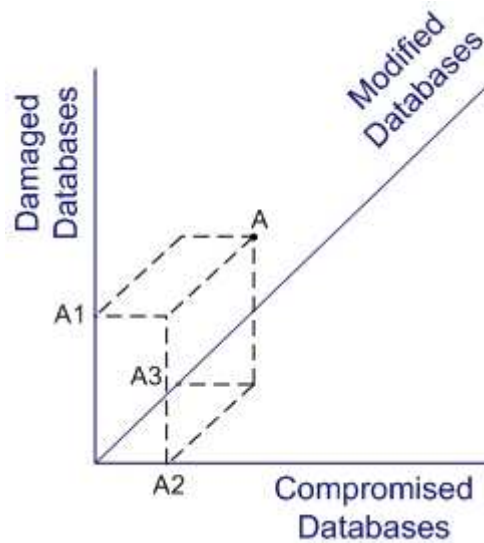


Figure 4.1. Orthogonality of the dimensions of database forensics based on [78]

For example, consider point A illustrated in Figure 4.1. This database which is being investigated has elements of all three dimensions of database forensics. The database is mostly a damaged database (A1) which has been modified since the event of forensic interest has occurred (A3). The database has been compromised in a lesser way and therefore the value of A2 is less.

The challenge of this orthogonal structure is to identify to which dimensions a database belongs before an investigation can take place. An idea of the dimension to which the database belongs may be extracted from the request that the forensic investigator has received in the first place. For example, if a company asks a forensic investigator to determine what has happened on a database that is not functioning anymore, the forensic investigator may guess that this database belongs to the damaged database dimension.

To determine beforehand if a database has been compromised is a challenge as well. Once again some information may be extracted from the original request to the forensic investigator. If the investigator is asked to investigate a database that is giving unpredictable results but is still in a working condition, the investigator may guess that the database is a compromised database. The modified database is a bit easier to determine because the investigator may have an idea of what has been done on the database after the

event of forensic interest has occurred. The investigator can achieve this by considering the environment in which the database resides. Ultimately, the true dimensions of the database can only be determined after running some tests on the database.

Another challenge is to quantify precisely how far a database resides on each axis. The value attributed to a damaged table versus a change in the code of the DBMS on an axis is wide open for debate. At the moment this is open for interpretation by the forensic investigator.

4.3 DATABASE FORENSIC TOOLS

In the previous section the field of database forensics is discussed in theoretical terms and information provided on the different types of databases that will require a forensic investigation. In this section useful tools that have already been developed are discussed. These may be used on databases during a forensic investigation. Several forensic tools exist to enable digital forensic investigations on various platforms such as EnCase, FTK and ProDiscover IR on file systems [92], Safeback on digital media, PeerLab on filesharing and forensic tools for smartphones [93]. There are many more digital forensic tools available; however, a discussion concerning digital forensic tools is not in the scope of this study.

There has been some development of database forensic tools but these tools often focus on incident response [84]. There are few tools that can help with incident responses on databases such as SQL Server Database Forensics for the SQL Server DBMS and Logminer for the Oracle DBMS. These tools only focus on one DBMS and on one dimension of database forensics. A database forensic tool that encompasses all dimensions of database forensics has not been developed yet. This tool should be able to analyse a database independent from the dimension of the database and the type of DBMS. This forensic tool should also be able to determine where the database lies on the orthogonal graph of the dimensions of database forensics [78]. A database forensic tool should also be proven to be accurate. The Oracle LogMiner was tested for potential use as a database

forensic tool, but due to some anomalies in the tool's functionality Wright [87] deemed the tool insufficient for use in database forensics.

David Litchfield, a database security guru and founder of Next Generation Security Software Company is in the process of developing a database forensic tool called Scalpel [94]. Once again, this tool is only focused on one DBMS called Oracle. The tool will automate the process of sifting through “mountains of system metadata” to discover the cause and extent of a database security breach. Later at a black hat conference, Litchfield said that “Right now the tools are all held together by bubble-gum and cello tape”, referencing the limited number of tools available to use for forensic investigations [76]. He also hinted that forensic investigators need to use different tools, which all provide a specific functionality, for one database forensic investigation.

The database itself could also be used as a database forensic tool [23]. The advantage of doing this is that evidence could be extracted from the tables and other structures of a database by making use of queries. The DBMS itself, as a forensic tool, is inadequate if the DBMS has been compromised. Imagine a situation where the code of the DBMS has been compromised in order to provide wrong results upon queries on a table. It is crucial to ensure that the DBMS is not compromised before the DBMS can be used as a forensic tool. If a DBMS is not compromised it can be a useful forensic tool.

4.4 FOUR ABSTRACT LAYERS OF THE DATABASE MANAGEMENT SYSTEM

As mentioned in an earlier chapter, database management systems enable users to access and store data without being concerned about the internal representation of databases. Data have to be interpreted to obtain logical information. A database management system consists of a collection of interrelated data and a set of programs to access the data [94]. The interrelated data can be raw data in files as well as metadata that are required to make sense of the raw data. The programs in the DBMS interpret the metadata and data in order to output useful information to the user of the DBMS. The DBMS is a complex concept with various segments that work together to deliver useful information to users. A simple definition of the DBMS has already revealed that a DBMS consists of data, metadata and

programs. The importance of dividing the DBMS into four abstract layers has been explained in a previous study [23].

In this section the various layers of a DBMS will be motivated and discussed in order to gain a better understanding of the interdependent layers of the DBMS. The layers will be selected in a manner that will make the selection applicable to forensics in any DBMS. Similar studies have been conducted to understand the complex layers of the DBMS and are discussed in a section to come. Thereafter another section will discuss why these layers are important in a digital forensic context. The various abstract layers will then be introduced, the hierarchy of the layers discussed and finally the boundaries between the layers described.

4.4.1 Background

In February 1975 the American National Standards Institute's (ANSI) Standards Planning and Requirements Committee (SPARC) published a report on the structure they had developed for DBMSs [35]. The study proposed that a DBMS consisted of three dimensions, namely the external dimension, internal dimension and conceptual dimension. An explanation of the dimensions was provided in the previous chapter. The complicated nature of the database warranted the database to be divided into three dimensions as early as 1975.

Olivier [23] envisaged the forensic advantages of dividing a DBMS into various dimensions or layers and making use of the various divisions during a forensic investigation. Some layers of the database might be dependent on other layers. For example, the application schema layer might be modified to influence the data layer to show incorrect results. In another example, integrity rules are typically stored in the data dictionary. To input data into the DBMS that does not conform to the integrity rules requires an attacker to change the data dictionary.

Lastly, in an example mentioned by Olivier, the attributes of an entity might be described in the data dictionary or application schema. An attacker could then change the attribute of

an entity to effectively delete the data or to swop data around. This meant that two columns such as *PurchasePrice* and *SellingPrice* could be swopped around for financial gain. This example will be illustrated in a later chapter.

The abovementioned examples show that there are interdependencies among the various layers of the DBMS that should be considered during a forensic investigation.

4.4.2 Why are abstract layers useful in a forensic investigation?

The question may spring to mind on why abstract layers can be useful in a forensic investigation. Several advantages of breaking the DBMS up into abstract layers for a database forensic study are the following:

- The DBMS is a complex software tool that is not yet understood in a forensic context. The various components of a database need to be understood in order to do a database forensic study.
- There are various layers in the DBMS which serve completely different purposes. The data of the DBMS are physical data presented as useful records. On the other hand, the DBMS consists of software functions which drive the workings of the DBMS.
- Layers can influence one another. A software function of the DBMS may be modified to not show a particular table in the DBMS. The interdependencies of the layers of the DBMS must be understood in a database forensic study.

4.4.3 Abstract Layers of the Database Management System

Output from databases is a function of the data it contains as well as the metadata that describe the data in the database. There are several levels of metadata which could manipulate the way in which data are perceived [23]. Since different levels of metadata influence the interpretation of data, it is similar to the problems that exist among forensic investigations on static data (dead analysis) and on live systems (live analysis). An analysis on static data is performed in a clean and reliable environment but there are several reasons why a dead analysis cannot always be done on static data [95]. A live analysis takes place in situ but with the possibility that the environment, like the operating system, can

manipulate the interpretation of the data. In a database it is necessary to achieve clarity on the levels of metadata and data for a forensic investigation.

In order to do some generalisation on the topic of database forensics, the various layers of the DBMS need to be viewed in an abstract way. This is required because of the need to construct an abstract view of DBMSs against which various combinations of metadata and data can be compared and studied. A DBMS consists of four abstract layers which can be grouped into a data model layer, a data dictionary layer, an application schema layer and an application data layer.

These four abstract layers of the DBMS are separated in order for the forensic investigator to narrow down a search for evidence in a case as well as to enable the investigator to harvest evidence from the database. An attack on a database can happen in any one of these four layers. An attacker can either change the physical application data, such as the amount of salary to be paid at the end of the month; modify the application schema, for example, to enable unauthorised users to access data that do not belong to the schema of that particular user; and change the data dictionary to possibly execute malicious tasks that influence the schema or data of the DBMS. The following sections will discuss each of the four abstract layers.

4.4.3.1 Data Model

The first abstract layer of the database is the data model which is a simple representation of complex real world data structures [96]. The data model is required by a database management system to enable the user to view the data in an understandable way. The data model should also be useful to the different types of users who make use of databases. These users include end users, application designers and designers of the database. The basic building blocks for all data models are entities, attributes, relationships and constraints while the building blocks of the data model are constructed and connected according to the design of the type of data model. In simple terms, the data model layer can be seen as the code that is written to assemble the DBMS. The data model is the code that dictates the workings of the DBMS.

The data model layer consists of various useful software tools which collectively work together to build the functionality that the DBMS should provide.

In a relational DBMS these data model layer software tools include but are by no means limited to the following:

- DDL and DML Processor. The data definition language and data manipulation language processors interpret commands that create, modify and delete database structures.
- Data Dumper. The function that is responsible for dumping data from a table is part of the data model layer.
- Syntax Checking Process. The syntax checking process checks that queries obey the syntax rules of the DBMS.
- Query Processor. A query processor checks if all the queried resources are available in the database and applies query optimisation to select the best query to run.
- Run Time Database Manager. This manager converts queries from the user's logical view to a physical file system view. The physical file system view then solves the query by interpreting the data and metadata of the DBMS.
- Database Login Procedure. The procedure that handles the login of a user into the DBMS is part of the data model layer.

In simple terms, the data model layer can be described as the part of the DBMS that is included in the source code of the DBMS. During the installation of the DBMS metadata are created that belong to other abstract layers. After installation the DBMS consists of a data model layer and other abstract layers. For example, the data dumper function exists in the source code of the DBMS and is hence part of the data model layer. After the installation of the DBMS the data dumper function is still part of the data model layer but now exists as object code in the installation. The data model has unlimited forensic possibilities from a forensic investigator's point of view. A list of various alterations or damages that could be applied to the data model layer has very limited use. In chapter 5 it is discussed how the data model could be investigated forensically.

4.4.3.2 Data Dictionary

The second abstract layer of the database is the data dictionary layer. The data dictionary layer can be described as the metadata which are relevant to all databases of the DBMS. In other words, the data dictionary layer is the metadata which will influence all databases of the DBMS if the data dictionary is modified. For example, the PostgreSQL DBMS has a data dictionary layer structure called `pg_database`. This structure is a table which contains metadata about how a database is configured. If the metadata in this table change, the configuration of the databases will change. The structure has the potential to change data of any database and therefore falls into the data dictionary layer.

A description of selected fields of the `pg_database` table is illustrated in Table 4.1. Several options exist to alter databases significantly within this table. Database names could be changed in the `datname` column, database owner information could be modified in the `datdba` column, access privileges could be swapped around in the `datacl` column. The DBMS is intelligent enough not to allow any information in a field of this table. Several integrity checks are executed to avoid incorrect data formats to be entered into a field. However, the attacker may find ways around the integrity checks in order to ultimately damage or attack the data dictionary.

In later chapters we will illustrate several examples of how the metadata of a DBMS could be attacked. Chapter 6 will illustrate how a forensic investigator should approach the data dictionary layer when conducting a forensic investigation.

Table 4.1. A description of selected fields of the `pg_database` table in PostgreSQL version 9.1 [97]

Name	Type	Description
<code>datname</code>	<code>name</code>	Database name
<code>datdba</code>	<code>int4</code>	Owner of the database, usually the user who created it
<code>datallowconn</code>	<code>bool</code>	If false, then no one can connect to this database. This is used to protect the <code>template0</code> database from being altered.
<code>datlastsysoid</code>	<code>oid</code>	Last system OID in the database; useful particularly to <code>pg_dump</code>
<code>datconnlimit</code>	<code>int4</code>	Sets maximum number of concurrent connections that can be made to this database. -1 means no limit.
<code>datacl</code>	<code>aclitem[]</code>	Access privileges

4.4.3.3 Application Schema

The application schema is the third abstract layer of the DBMS. The application schema can be described as the database designer's database, as it records the design decisions about tables, views, indexes and other database designer structures. The application schema includes all metadata relevant to a *single database* in the DBMS. The data dictionary layer's metadata are independent of a database, whereas the application schema layer's metadata are dependent on a single database. For example, the application schema contains metadata about the tables created by a user in a database [96].

The idea of an application schema consists of the respective parts:

- An indication of which data can be manipulated by what user or application instance is stored within the application schema.
- The actual operations that are created to manipulate data form part of the application schema. These can include but are not limited to database triggers, procedures, functions and sequences [98].
- The logical grouping of database objects such as views, indexes and tables forms part of the application schema [96].
- Metadata tables which are relevant to a single database are included in the application schema layer. For example, the `pg_aggregate` table in PostgreSQL consists of all metadata relevant to the aggregation functions available to a database. A copy for this table exists for each database in the DBMS. Aggregation functions could be modified by changing the values in this application schema structure.

The application schema is part of the database that is known by more database users as opposed to the data model which may be known well by database development experts only. Database administrators and database designers work on the application schema layer (or parts thereof) on a daily basis. The magnitude of activity on the application schema as well as the large number of users who understand the application schema layer present forensic ramifications on the application schema level. These ramifications of forensics on the application schema layer are discussed in chapter 7.

4.4.3.4 Application Data

The last abstract layer of the DBMS is the application data layer. The application data refer to the actual data that are stored logically within tables in the database and physically within data files on the database server. Logically, application data could be accessed by running queries on a table and the application data are returned as records consisting of various fields of data. The data model, data dictionary and application schema layers of the DBMS contribute to the process of presenting physical data stored in data files on the database server as database records. The application schema is arguably the most important part of the DBMS from a DBMS user's perspective.

The other layers of the database are designed in order to present the application data to the user in an understandable and usable format. The application data layer is the layer which might be the most critical in a forensic investigation due to the fact that, in most cases, the application data would be the ultimate target of an attack on the DBMS. To name a few examples, attackers might want to destroy a data dictionary structure in order to ultimately destroy data; an attacker might also modify the application schema in order to ultimately modify the way in which the application data are presented. Finally, an attacker may change the application data directly in order to commit fraud. This means that the application data will be targeted in most attacks on the DBMS. However, the attack might happen on any of the four abstract layers of the DBMS.

4.4.4 Hierarchy of Abstract Layers

The four abstract layers of the DBMS are not all equal. There is a hierarchy that exists among the four abstract layers of a DBMS. This is due to the inherent nature of how the DBMS was designed. Figure 4.2 displays the hierarchical structure of the abstract layers of the DBMS. The data model layer is at the top of the hierarchical structure because of the significant influence it has on the other three abstract layers of the DBMS. Remember that the data model is the code of the DBMS. A change in the data model layer could change the way in which the metadata of the application schema and data dictionary are interpreted. A change to the data model layer could also alter the way in which application data are presented to a user.

The data dictionary is second on the hierarchical structure of the abstract layers of the DBMS. The data dictionary layer depends on the data model and cannot influence the data model. However, the data dictionary could modify the way in which the metadata of the application schema are interpreted or modify the way in which the application data are displayed to the user. For example, a database level setting might change in the data dictionary which would cause the tables in the application schema layer to change location, disappear or be inaccessible by DBMS users.

The application schema layer is influenced by the data model and data dictionary, but could influence the application data. For example, a change to an application schema table structure could delete application data from a table.

At the bottom of the hierarchical structure of the abstract layers of the DBMS sits the application data layer. This layer is arguably the most important layer of the DBMS but is influenced by all other layers higher up in the hierarchy. The application data are thus a function of the data model code, data dictionary metadata and the application schema metadata.

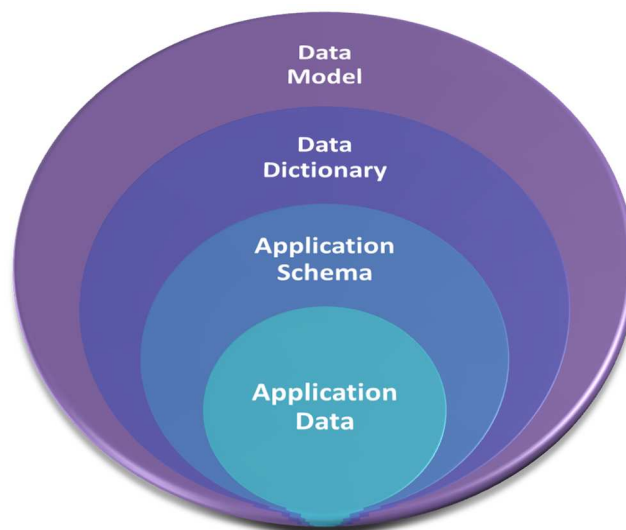


Figure 4.2: Hierarchical structure of the abstract layers of the DBMS

4.4.5 Boundaries Between Abstract Layers

This study is intended to be applicable to a variety of relational DBMSs and therefore, the boundaries between the four abstract layers of the DBMS are not always precisely the same for all DBMSs. It is not the intention of this study to generate a list of boundaries between abstract layers but rather to demonstrate the logic behind the boundaries of the abstract layers of the DBMS. Examples of boundaries will be illustrated by focusing on the PostgreSQL DBMS but the same logic should be applied to all relational DBMSs in order to find the dividing lines between the abstract layers. The boundaries between the abstract layers are important. Hence, when we want to focus our investigation on a single layer of the DBMS, we need to ensure that we are in fact working with metadata or data in that particular layer.

Boundaries could be discussed at two logical levels in the DBMS. The first level is the boundaries between the physical files installed on the database server which will be called the *physical level* in the remainder of this section. The second level is the boundaries in the user interface of the DBMS which will be called the *interface level* for the remainder of this section. The following sections will discuss the boundaries between the four abstract layers of the DBMS. The practical use of these boundaries will be illustrated in chapter 8.

4.4.5.1 Boundary Between the Data Model and the Metadata Layers

The boundary between the data model and the metadata layers will be different in various databases. The *metadata layers* refer to the data dictionary and application schema layers which consist of metadata to help construct databases and their structures. On the physical level, the forensic investigator should have enough knowledge about the investigated DBMS to be able to determine which installed files on the database server are part of the code of the DBMS or are code functions that form part of the data model. There will also be files on the installed database which contain the metadata of the DBMS. For example, the `pg_dump` file which is a function that could be executed in order to process a dump of the application schema and application data. This `pg_dump` function that could be executed forms part of the data model layer. The PostgreSQL DBMS has a *data* folder which

consists of all the metadata and data of the DBMS installation. The data in these files belong to the other three layers of the DBMS.

4.4.5.2 Boundary Between the Data Dictionary and Application Schema

The boundary that separates the data dictionary layer and application schema is the fact that the data dictionary layer is metadata which are independent of a database and the application schema layer is metadata which are dependent on a database. The logic behind the dividing line between the data dictionary and application schema is explained in detail in chapter 6. On a physical level, the layers may be divided among various files or they may reside in the same file. The boundary between the application schema and data dictionary on the physical level depends largely on the DBMS design. On the interface level it is easier to distinguish between the data dictionary and application schema. As mentioned earlier, the `pg_database` metadata table includes data which are independent of a single database and therefore part of the data dictionary. Metadata tables, such as `pg_class` and `pg_aggregate`, are relevant to a single database and form part of the application schema layer. The metadata tables could be accessed via the interface on a PostgreSQL DBMS.

4.4.5.3 Boundary Between the Application Schema and Application Data

The boundary between the application schema and application data divides the physical level and the interface level. On the physical level, the layers may be stored in different files, which is the case in the PostgreSQL DBMS. Furthermore, the boundary between the application schema and the application data could be viewed in a dump file of a table. For example, the *create* SQL commands in a dump file would refer to the application schema structures and the *insert* SQL commands would refer to the application data which belong to a table. On the interface level, the boundary between the application schema and application data could be interconnected but could be viewed separately. The application data could be viewed as records of a table by running a select query on a table. The application schema could be viewed in a MySQL DBMS by making use of the *explain* command.

4.5 CONCLUSION

In this chapter the field of database forensics was explained. It was also illustrated that there is a significant shortage of literature in the field of database forensics. It was then illustrated that the DBMS could be divided into four abstract layers. This chapter has prepared the reader for the discussions to follow in the next chapters because the differences between the various abstract layers are explained and the current state of the database forensic field is discussed. The abstract layers will provide a basis on which this study will build. Within chapter 4 the attention has shifted from existing work to the contributions of the authors towards the field of database forensics.

The following chapters will discuss each abstract layer in a forensic context and provide solutions to the forensic challenges faced on each abstract layer of the DBMS. As mentioned before, each of the extensional abstract layers is discussed individually because they are so different from one another and an understanding of the layers in a forensic context is required in order to solve the problem of this study – which is to find an appropriate combination of the forensic states of the various abstract layers in order to conduct a forensic investigation on an entire compromised DBMS.

CHAPTER 5 DATABASE DATA MODEL FORENSICS

The previous chapter discussed that a database management system (DBMS) consists of metadata and data. The metadata influence the way the data are presented to the user and this presents various forensic complications. The data model can be viewed as the highest level of metadata which governs the way in which other metadata and data in the DBMS are presented to the user. The data model can be modified to hide or tamper with forensic evidence.

In this chapter the focus is on the data model of the DBMS and arguments are provided to indicate why the data model is an important consideration when conducting a forensic investigation on a DBMS. Various methods are presented to transform the data model into a desired state for a forensic investigation and these methods are measured against set out criteria. No one method is adequate for every forensic investigation. A forensic investigator should understand the various methods, and select the correct data model state and the method to convert the data model into that required state. The concept of a forensic study which focuses on the data model was published as part of this study [101].

5.1 BACKGROUND

A database management system (DBMS) consists of various layers which can either be a form of metadata or data [99]. The ANSI-SPARC Architecture divided the database into various layers [100]. Some layers in the DBMS may influence the actions and results of other abstract layers, and this fact presents a variety of vulnerabilities and forensic possibilities [23]. Based on the ANSI-SPARC Architecture, we divided the metadata and data of the DBMS into four abstract layers in chapter 4. The data model abstract layer of the DBMS is a type of metadata and can be viewed as the code of the DBMS. The other abstract layers of the DBMS include the data dictionary, application schema and application data.

In this chapter the focus will fall on the data model abstract layer (source code) of the DBMS, because it can be viewed as the highest level of metadata which influences other

metadata and data in the DBMS. The data model of a DBMS interprets the metadata and data stored in the DBMS, and delivers the results accordingly but the data model is the source code which can be altered to deliver almost any result. In another paper published as part of this study, the data model can be changed to tamper with evidence, such as table structures, records in a table etc. [102]. Similar to any other software program, the source code of the DBMS may present some flaws and vulnerabilities [2, 8, 103].

Our premise is that the ideal tool to examine the contents of a database will often be a DBMS: it provides powerful query facilities that enable the investigator to express the wish to easily formulate exactly what data are of interest. Possibilities exist to correlate sets of data or to find data that are inconsistent with other data. Possibilities exist to extract data that conform to a complex list of constraints. Possibilities exist to formulate simple queries just like a typical user would have done it. Possibilities exist to query the metadata to determine the structure of data. And so on. However, in order to use the DBMS as a tool for such forensic database examination we need to know that we can rely on it to provide forensically useful results.

Formulated slightly differently, the challenge is to combine the data to be investigated with a DBMS that is deemed reliable enough. Note that *reliable enough*, in general, means that the tool being used should exhibit sufficiently low error rates. The phrases *sufficiently low* or *reliable enough* are, amongst others, influenced by whether the case at hand is a criminal or civil matter, and whether the outcomes are to be used for evidentiary purposes or some lesser role in an investigation. Also note that the *data to be investigated* mentioned above may include metadata below the data model in addition to application data on the lowest layer.

This chapter examines various ways in which a DBMS may be combined with data to be examined. It assumes the data to be examined are located on some computer that potentially contains evidence. For such data to be examined, only two alternatives exist: either the data are examined on the computer where it is located or the data are somehow imaged and copied to a forensic machine. More options exist for the DBMS to be used as a

forensic tool, from using the DBMS as found on the computer with the evidence or using this DBMS as imaged from this computer, and then restored and run on a forensic machine to installing a pristine copy of the DBMS on the forensic machine, combining it with the data to be examined. This chapter works systematically through the viable options, examines how the option may be effected in an examination and determines the reliability of such an option in qualitative terms.

These viable options will have to fulfil one or more of the following criteria:

- Provide an environment where the evidence output from the DBMS is trustworthy and has not been altered by the data model.
- Provide an environment where the processes used to acquire consistent evidence from the DBMS have not altered the evidence.
- Provide an environment where a forensic hypothesis can be tested on the DBMS.

In order to discuss the various methods and whether they adhere to the criteria mentioned above, this study is structured into the following four key areas:

1. The various reasons why either a found data model or clean data model is important are discussed. This section is included to familiarise the reader with the differences between a found and clean data model, as well as discuss the advantages and disadvantages of a found and clean data model.
2. In the next section a range of methods are presented to achieve a clean or found data model environment. Various methods can be used to achieve either a clean or found data model environment.
3. The forensic soundness, cost, practicality and the adherence of these methods to our criteria are discussed.
4. The final section explains how this study fits into the bigger picture of database management system forensics and the remaining three abstract layers.

5.2 FORENSIC ENVIRONMENT WITH EITHER A CLEAN OR FOUND DATA MODEL

The previous section briefly discussed how the data model may influence the output of the DBMS. Two types of data model forensic environments will now be explored: the clean and found environment. The difference between these two environments is that the found environment contains the original data model that was present when the incident of forensic interest occurred, while the clean environment represents an environment where the data model has been cleaned up in such a way that we can trust the data model. The found environment can further be divided into a found environment that either resides on the same machine or a found environment that is copied onto a completely different machine. In this section the various reasons why the found or the clean environments are useful will be discussed. Several scenarios are presented where the found or clean environments are advantageous.

5.2.1 Arguments for a Clean Data Model Environment

The clean environment is a setting where we have ensured that the data model will not alter the output of the DBMS. It is important to understand that a clean state differs from a post-mortem state characteristic of traditional digital forensics. A clean state is not merely a copy of the evidence that needs to be analysed, but rather a copy of the evidence program (or DBMS in this case) that runs like the original copy and from which we can query output. This means that the clean environment is set up to run like the original DBMS but we are sure that the data model is not corrupting the output that we receive from the DBMS. But why would it be important to modify a data model into this clean state? The various reasons why it can be important consist of the following: (1) the data model as source code, (2) the data model as a rootkit, and (3) the data model as a toolkit.

5.2.1.1 The Data Model as Source Code

The data model is the underlying software that controls the way in which the metadata and data of the DBMS are utilised. Just like any other source code, the data model might have the ability and authority to control the way in which the metadata and data of the DBMS are stored, interpreted and processed. Given that the data model is the source code of the

DBMS, we can imagine that the source code can be changed in endless different ways to achieve almost any outcome. Depending on the DBMS, the source code may be freely available on the Internet. An attacker can edit the source code to execute the attacker's intended commands before installing or updating the compromised code on a host or server. For example, the attacker can edit the source code to always keep a record of a beneficiary in a salaries table; the attacker can rewrite the dumper of the DBMS to hide certain data when a dump is made; the attacker can rewrite the source code to reveal private information of other individuals in regular intervals etc. Due to the fact that the data model can be rewritten to execute malicious tasks, hide data etc. it is important to ensure that the data model is clean when doing a forensic analysis.

5.2.1.2 The Data Model Rootkit

In this section we discuss the second reason why a clean data model state could be required. Rootkits may replace programs or executable files on a host to execute a different task than the executable file was originally intended to execute [104]. Rootkits are well-known to attack operating systems but it is possible that a rootkit may attack a DBMS. The database is very similar to an operating system because both have users, processes and executables.

Red Database Security [105] illustrated how the oracle database could be attacked by a rootkit. Users were hidden in the database by inserting a username in the users table and then rewriting rules of a view to not display the inserted username. The view name in oracle was sys.all_users and then changed to system.all_users. It was set up to refer to the original view but concealed some users. In a similar fashion the database processes were hidden by changing the view name of where the processes were stored from vsession to v_session. The new view points to the old view but filters out all the processes that we do not want the normal users to see.

An intruder can install a rootkit on a DBMS by acquiring access to the database through default password guessing, TNS Listener Exploits, Operating System Exploits etc. The example of installing a rootkit in an oracle database is illustrated in [105]:

1. Create a file `rootkit.sql` on your own webserver.
2. Insert a HTTP-call into the `glogin.sql` file on the attacked DBMS client.
3. The next time the database administrator logs into the attacked DBMS the `rootkit.sql` is downloaded from the webserver. Then the `rootkit.sql` file is executed to: disable terminal output; create a user; modify data dictionary objects; download and execute a keylogger (`keylogger.exe`); or do any other malicious task.
4. Finally the terminal output is enabled again.

A large variety of executable files can be uploaded to the DBMS and a situation is created where the DBMS cannot be trusted. A clean installation of the data model is required to ensure that the forensic evidence collection process will not be influenced by a rootkit.

5.2.1.3 Data Model as a Toolkit

Lastly, we have a look at how the DBMS and its data model can be used as a toolkit. This is the third reason why a clean data model state could be required. Forensic analysts often use toolkits to extract evidence from a device. Toolkits need to ensure that the evidence collection process does not alter the evidence at the same time. The EnCase toolkit brought about a revolution in the digital forensic arena when it introduced the concept of a toolkit that mounts the bit-stream forensic images as read-only virtual devices; therefore, disabling any alteration of evidence on the digital device [106].

We argue that the data model of the DBMS can also be considered as a type of digital forensic toolkit to extract evidence from the DBMS. While an operating system forensic toolkit extracts registry entries, retrieves deleted data or validates partitions from the operating system, the data model acts as a forensic toolkit by providing records in tables, logging information and reporting database structures. Therefore, it is important that the data model is in a state where the output it delivers can be trusted. If the data model is used as a digital forensic toolkit, it should adhere to the requirements of a digital forensic toolkit.

A toolkit should be [107]:

- Definable. One must be able to state the problem, describe the desired outcome, develop the algorithm and validate the process.
- Predictable. The tool must be designed to function in a predictable manner across any usage of the tool.
- Repeatable. The function must be repeatable within a small error range.
- Verifiable. The output must be able to be verified in various toolkit testing environments.

In the case of a compromised data model it can be assumed with certainty that the compromised data model will not be predictable and we cannot deem the compromised data model to be repeatable because we cannot predict what output a compromised data model will return next. On the other hand, when we have a clean data model we can say that the data model will be definable, predictable and repeatable. It is more complicated to say that even the clean data model is verifiable, because the data model is actually our sole verification of what evidence exists in the data files. A data dump might help us to verify some output but this is inherently part of the data model. We argue that the data model can be used as a digital forensic toolkit when the data model is clean.

5.2.2 Arguments for a Found Data Model Environment

A found environment in the data model of the DBMS refers to a situation where the data model that has been in use in the DBMS when an event of forensic interest has occurred is utilised. The found environment may also refer to an environment where the same machine where the DBMS has been installed originally is not used, but the data model has been mirrored from the original machine. It is vital to understand that the found environment is not exactly 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 same machine or on another machine. In this section, reasons are discussed why there may be a need for a found data model environment when conducting a DBMS forensic investigation consisting of: (1) cost of forensic investigation, (2) testing a hypothesis, and (3) simple queries.

5.2.2.1 Cost of Forensic Investigation

A vital consideration before conducting a forensic investigation is to determine how great the cost of the forensic investigation will be. Each technique of conducting a forensic investigation has a different level of complexity and different resource cost [108]. These resource costs include hiring the experts to conduct the investigation, the duration of the investigation and the tools that the experts require to complete the investigation.

The cost of the forensic investigation applies to the DBMS. For example, the cost of the expertise to replicate a DBMS in a forensically sound manner to achieve a clean data model should be considered. To achieve a clean environment takes a lot of time, effort and specialised expertise. Not all forensic investigations can afford the processing time and cost of this type of forensic investigation. A found investigation can be a very cost-effective investigation where the forensic investigator can log into the database and use simple database queries to extract all the evidence that is required. On the other hand, it may be a waste of time and resources to acquire useless evidence from the DBMS by taking a cost-effective approach. It may be more cost-effective to conduct the investigation on a clean data model from the start. Nevertheless, the first reason why an investigation of a DBMS with a found data model can be useful is due to its lower cost.

5.2.2.2 Testing a Hypothesis

The second reason why a found data model might be useful for a forensic investigation is to test a hypothesis. A hypothesis is an informed supposition to explain the observations in the data gathering phase of a forensic investigation [106]. In other words, a hypothesis is a theory which an investigator may have about what has occurred and this theory needs to be tested. A hypothesis may be wrong at first but it is a good place to start when gathering evidence from a digital device. A hypothesis needs to be tested to confirm if it is correct. A hypothesis may need to be tested in the DBMS but there is no need to change the data model environment in order to test the hypothesis. In fact, it is better to test a hypothesis when we know that the data model and underlying code are in the same state when the event of forensic interest has occurred. In this instance, a found data model will be required to test the hypothesis.

5.2.2.3 Simple Queries

The third reason why a found data model can be useful during a forensic investigation is to run simple queries that will have no noticeable effect on the operations or state of the DBMS. The results that the found data model gives as output can still not be trusted entirely but in some cases it may not be that critical to replace or cleanse all the code of the data model. There may be a situation where the investigator needs to search for clues and not forensic evidence. Examples of such situations include a quick check if a user logged in regularly; checking how many rows there are in a table (with a margin of error due to the fact that we cannot trust the data model); using the DBMS as a mechanism to double check some results that have been acquired from another source; making use of database queries to build up a hypothesis etc. Only a found data model can provide the forensic investigators with such quick access to data but here it is very important that the investigator knows that he cannot trust the results entirely.

5.3 HOW TO ACHIEVE VARIOUS STATES OF THE DATA MODEL

Thus far this chapter has discussed why there is a need to do a forensic investigation on either a clean or found data model environment. In this section we will discuss various methods on how to achieve a clean or found data model. To make changes to the state of a data model is a complicated task and to achieve a clean state might be the most difficult of all. On the other hand, it may become complicated to achieve a desired found state, especially if we want to realise a found state that is a hybrid of found data and some clean data elements.

5.3.1 Clean Environment

To transform a data model into a clean state requires that the data model be transformed into a condition where the forensic investigator can trust the data model and be certain that the output of the DBMS is in no way corrupted by the data model. To achieve a clean state often requires rebuilding the DBMS because the data model is such an integrated part of the DBMS. In short, the data needs to remain the same but the source code needs to be replaced or cleansed.

A critical requirement for the transformation of the data model to a clean state is that the process has to be forensically sound and does not corrupt the data model even further. The clean state of the data model can be a less forensically sound condition if the process to transform the data model into a clean data model is not forensically sound. Various methods to achieve a clean data model are discussed in this section consisting of: (1) copying data files, (2) recovering the DBMS, and (3) patching the DBMS.

5.3.1.1 Copying Files

One of the approaches to achieve a clean state of the data model is to copy the data files from the suspect installation of the DBMS to a new installation of the DBMS where a clean copy of the DBMS has been installed. The logic here is that a new installation of the DBMS (where the installation files have been retrieved from a trusted source) on another machine will provide a clean data model. The data files of the new installation will either not exist or will have no data in them. The data files of the new installation will be replaced with the data files of the suspect installation. A clean data model along with the data that exist on the suspect machine will then be achieved.

The process to achieve this clean state by copying data files is started by determining where the dividing lines of the four abstract layers of the DBMS are within the DBMS installation files. The data model needs to be separated from the other three abstract layers in the files of the DBMS installation, because it is necessary to ensure that no part of the suspect data model is copied to the clean environment.

Once the dividing lines between the data model and the rest of the abstract layers have been determined, all the files need to be copied and the md5 or other hash function of these files needs to be calculated in order to ensure the consistency of the data.

Next a clean installation of the DBMS needs to be installed on another machine. The identified files can then be copied into this clean installation after the files have passed another md5 check on the new machine. All identified files should be copied in the new DBMS installation into their correct directories. The new DBMS installation with the new

data model should now be ready for a forensic investigation and the investigator should run some tests to confirm that the DBMS is in a working condition.

This process has been tested with positive results on a PostgreSQL DBMS in one of our studies [102]. This copy process is illustrated in Figure 5.1.

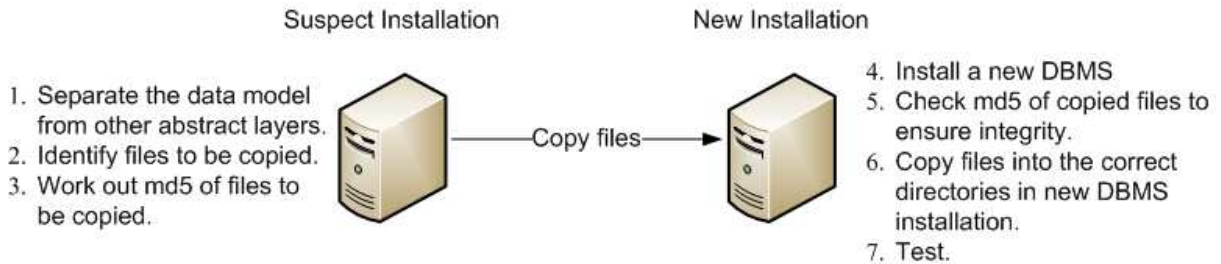


Figure 5.1. The copy process to achieve a clean data model

This process can be deemed forensically sound if the dividing lines between the data model and the rest of the abstract layers can be defined, and it can be proven that no data model data are mixed between the files to be copied. The md5 calculation helps to prove that the data that are copied from the suspect machine are the same as the data that are copied into the DBMS installation on the new machine. It becomes difficult to achieve a clean data model by copying files when the DBMS has been patched various times and has had various add-ons installed. These patches and add-ons need to be replicated onto the new DBMS installation.

This approach of copying data files over to accomplish a clean data model state is practical and has been proven to be possible to do but it is difficult to determine the dividing lines between the data model and the other abstract layers. The dividing lines will also be unique for each type of DBMS. To apply all patches and add-ons may also be a tedious task but something that is possible. This method adheres to two of the criteria mentioned at the start of this chapter. The data model will not alter the evidence and the evidence can be trusted. It does not make sense to use this method to test a hypothesis, because a hypothesis can be tested on the found data model without any effort to convert the data model to a clean state.

5.3.1.2 Recover DBMS

Another approach to achieve a clean data model state in the DBMS is to recover the DBMS to a previous state with backup data. If we have adequate backup data of the DBMS we can recover the DBMS to a state that is very similar to the state of the suspect DBMS but with a clean data model.

This method starts by determining if we have satisfactory backup data to recover the DBMS into an acceptable state. It is necessary to ensure that the application schema and the application data abstract layers are recovered in order to correctly accomplish a clean data model which is still acceptably similar to the suspect DBMS. Old data dumps of the entire DBMS will work particularly well in this instance, because data dumps will include the application schema (like tables, view and trigger structures) as well as application data (the records or data within in the tables) if the data dump was made before the DBMS was corrupted.

Another fine source of recovery data will be a backup server for the DBMS where it can be proven that the backup server has not been compromised and we can trust the data model [109]. This backup installation will have its own data model but the application schema and application data have been copied to the backup server at every specified timeframe (like every midnight). If we do not have the luxury of a backup server or any kind of data dump it becomes difficult to recover the database. The application schema can then be rebuilt by making use of database design documentation if such documentation exists. However, the application data will need some source of records of the data that were stored in the suspect DBMS, like spread sheets or any other source.

Once we have the acceptable amount of backup data, we need to install a clean copy of the DBMS onto another machine. The installation should be exactly the same as the suspect DBMS, and all patches and add-ons should be updated and installed. The application data and application schema can then be imported to the new DBMS installation by either entering the data through the DBMS console interface or by making use of the DBMS's data import tools. Unlike the Copying Files method, this method makes use of the clean data model to import the data into the clean DBMS installation.

If the investigation requires that the data be as similar as possible to the suspect DBMS's data it may be useful to run any scripts on the data that would have run on the suspect machine. The Cron tool in Linux is known to list all the processes to run on the machine each specified timeframe [110]. If Cron or a similar technology is used, the scripts that have been run on the suspect machine can be determined, and we can run the same scripts on the new machine in the correct sequence. After the data have been imported we can start with the forensic investigation on the clean data model.

This is a practical approach to accomplish a clean data model, because it is common practice to make regular backups of the DBMS, either to a FTP server or a backup server with a DBMS installed on it. This approach is useful when something has gone wrong on the DBMS and it can be determined when this event has occurred. It is then possible to recover the DBMS back to the date before this event has occurred.

This approach can also be useful when not all of the data of the database need to be recovered but it is just necessary to run tests on one isolated table. Only the application schema and application data of the one table need to be recovered. On the other hand, it may be difficult to determine when a compromise has occurred and various backups need to be attempted before the desired results are achieved. Furthermore, when there were no or little backups made of the suspect DBMS, it becomes very difficult to recover the DBMS.

Lastly, this process is forensically sound since the DBMS data is recovered by making use of very common and thoroughly tested techniques of recovering data. This method adheres to at least one of the three criteria, because the data model can be trusted and if the outdated backup data has no effect on the evidence, this method will also not alter the evidence of the DBMS.

5.3.1.3 Patching the DBMS

Another method to achieve a clean data model is by patching the DBMS. A DBMS patch is software designed to fix problems or update the software of the DBMS [111]. A patch

has the potential to update the data model of the DBMS, and thus provides a clean and trusted (or at least a partially trusted) data model by merely running a patch script. The patch is a trusted source of code. If this code can replace the untrusted data model code a clean or cleaner data model may be achieved effortlessly.

The two scenarios when this method can be utilised are to clean up the entire data model to acquire trusted results or to fix odd results that are retrieved on the suspect DBMS by cleansing the data model with a patch. In the first scenario it needs to be determined if there is a patch for the relevant DBMS that will clean up the entire data model without influencing the evidence within the DBMS. Patches will specify precisely what parts of the DBMS the patch will update in the patch readme and the extent of the data model clean-up can be determined from the specification.

In the second scenario the suspect DBMS gives unexpected output that could be caused by various reasons, such as hidden data, malicious code etc. A patch may be applied to attempt to eliminate the incorrect output and reveal new evidence. If changes have occurred between the suspect DBMS's output and patched DBMS's output, the patch readme document will provide us with a focus area to determine where in the data model the problem has occurred.

The investigator needs to decide if the patch will need to be applied on the same machine or on a different machine. To apply the patch on the same machine where the suspect DBMS is installed may not be a popular idea, because it will seldom be acceptable for the owner of the DBMS that an investigator patches the DBMS and runs the risk of possibly damaging the DBMS. It can therefore be argued that the patch needs to be applied to another machine in order to create a forensic environment with no risk to the owner of the DBMS.

The DBMS must be installed on the other machine with the same version and add-ons as the original DBMS in order to make sure the installation creates all required configurations on the new machine. Then the files of the entire original DBMS need to be copied onto the

new machine. The md5 check must be executed to ensure that no data have changed in the copy process from the original machine to the new machine. The copied files of the original DBMS must now replace the files in the new installation. Finally, the selected patches should be applied to the new DBMS installation. The new DBMS copy can now be tested and the investigation can start on the clean DBMS.

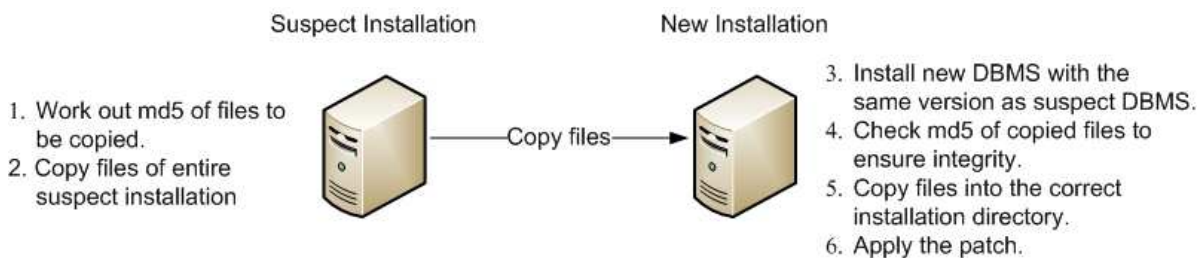


Figure 5.2. The patch process to achieve a clean data model

An advantage of this method is that we can find an almost exact copy of the suspect DBMS, because we do not have to revert to a backup. A disadvantage of this method is the fact that it may be difficult to prove that the patch will clean up the entire data model. This will require knowledge of the specific DBMS from the forensic investigator to determine precisely what part of the DBMS will be patched. The process can be forensically sound if it can be proven that the patch cleans up the data model to an extent that is required for the relevant investigation. This method adheres to two of the criteria. The data model can be trusted and the evidence will not be altered. Although we can test the hypothesis that the patch will fix a certain part of the DBMS, this method does not provide a way to test various hypotheses.

5.3.2 Found Environment

The previous section discussed the various methods that can be used to achieve a clean state of the data model which will be acceptable for a forensic investigation. In this section the discussion will include methods to achieve a found data model environment which is acceptable to conduct a forensic investigation in. A found data model involves that the original data model of the DBMS be utilised to conduct a forensic investigation.

As mentioned before, the data model cannot be trusted when conducting an investigation on a found data model but there may be situations where this kind of an investigation may be useful or practical. The various methods that exist to do an investigation on a found data model consist of: (1) an on-site investigation, (2) replacing fragments of the DBMS, and (3) backing up the DBMS.

5.3.2.1 On-site Investigation

The on-site investigation involves doing an analysis on the original DBMS where an event of forensic interest has occurred. The on-site analysis provides the investigator with access to the DBMS without doing much preparation of an environment. The approach to achieve an on-site investigation is a simple method that only requires the investigator to organise access to the DBMS and a timeframe allocated to conduct the forensic investigation. Traditional live investigation practices need to be applied here and the investigator needs to be mindful not to corrupt or change the evidence. It is good practice to back up the DBMS before starting the investigation, and to make a copy of the log before and after the investigation so as to record what the investigator has done during the analysis.

This approach gives the investigator quick access to potential evidence; neither does it require a list of tasks and specialised expertise to prepare the correct investigative environment. One drawback of this method is that we are working on an untrustworthy data model and it may be wrong to use the output of the DBMS as evidence. The output can be a false picture painted by injected source code or rootkits on the DBMS. Another drawback of this method is that operations may need to be halted while the forensic investigator is working on the DBMS which may cost the parties involved a lot of money.

This method adheres to at least one of the criteria: a hypothesis can be tested in this found environment. Care should be taken to not alter the evidence. Although the data model cannot be trusted, there may be instances where this fact will not halt the forensic investigation.

5.3.2.2 Replace Fragments of the DBMS

Another approach to conduct a forensic investigation of a DBMS with a found data model is to replace fragments of the DBMS installation on the suspect machine in order to realise a more trustworthy data model. In this context, replacing fragments refers to the replacing of executable files or other files that belong to the suspect data model with a similar but trusted file. The forensic investigation will be conducted on the live data model while parts of the data model will have been replaced in order to obtain a more trusted data model. If tests can be conducted to determine which part of the data model cannot be trusted, it is possible to replace that fragment of the data model with this method.

The first step in the process of utilising this fragment replacement method is to determine which part of the data model has been compromised. An md5 or other hash can be calculated on the files of a new clean data model installation and then compared to the md5 of the files on the found data model. If the hash is found not to be the same on the two DBMS installations, then a possible fragment to replace has been discovered. If this file can be linked to the data model of the DBMS and the replacement of the file will have no effect on the data or schema of the DBMS, the file can be approved for replacement. Of course, the investigator will need a thorough understanding of the relevant DBMS, the dividing lines between the data model and the other abstract layers, and the files that are installed in the DBMS installation or what the purpose of the file is. Once the investigator has found files to replace the corrupt files from a clean data model, the investigator needs to backup the current files first so that the DBMS can be rolled back if something goes wrong. Thereafter the files can be replaced by the new clean files. Hence, this results in a found data model which can be trusted.

This method utilises some advantages of an investigation on a found data model and provides the additional benefit that we can trust the data model. If the corrupt activities of the data model can be narrowed down to a couple of files, it may be easy to swap the files with clean files from the source code of the DBMS. The whole data model can be replaced in this way, given that the investigator knows precisely what the data model of the relevant DBMS is and if there are files that can easily replace the corrupt files.

This approach is risky to the owner of the DBMS, because a change to the installation files of the DBMS can crash the DBMS. This method will be heavily reliant on the expertise of the investigator to ensure that the correct fragments are replaced and that the DBMS is not damaged. This approach adheres to at least two of the criteria, if applied correctly. The data model can be trusted and the evidence will remain unchanged.

5.3.2.3 Backup the DBMS

Imagine a situation where a DBMS server with data produces a lot of revenue to a company. It is standard practice to backup the DBMS to protect the company's revenue from a DBMS failure. When conducting a forensic investigation a situation may arise where the original DBMS has been compromised but the backup DBMS on another machine or medium is still functioning normally and is not affected by the compromise. The original DBMS has a compromised data model that cannot be trusted which points to all our data and the backup DBMS has a trusted data model with all the same data.

This approach can be achieved by confirming that all data are copied to the DBMS by looking at the scripts that copy over the data and confirming that the copy has happened by checking log files and actual entries. It is also important to determine if the backup DBMS has not been affected by the compromise and the DBMS of the backup can still be trusted. Some log files may reveal the compromise that has occurred in the original DBMS, and then the backup DBMS's log files can be checked to see if the same compromise has happened there.

This is a simple approach if granted the luxury of a backup DBMS that has not been compromised. The reason for this is that we have a trusted data model with all the same data. The concern here is how forensically correct it is to conduct a forensic investigation on a backup copy of the evidence. When using the backup DBMS for forensic evidence it must be proven that the processes used to make the backup copy of the DBMS are just as forensically correct than making a copy of the original DBMS in a forensically correct way and taking the evidence into the laboratory. It should also be proven that no user has edited

the data on the backup server. Essentially the backup server can then be perceived as a copy of evidence where the data model has been cleaned up. Furthermore, the backup DBMS can at least be used to assist in the current forensic investigation, because the data model of the DBMS may still be in a better state than the data model of the original DBMS. This method adheres to one of the criteria, because one can trust the data model.

5.4 DATA MODEL FORENSICS IN CONTEXT WITH THE OTHER ABSTRACT LAYERS OF THE DBMS

Various reasons to justify the need for a clean or found data model when conducting a forensic investigation have been discussed as well as how these forensic environments can be set up. It is now important to consider how this discussion of the data model fits in with the rest of the abstract layers of the DBMS. The DBMS consists of four abstract layers which are known as the data model, data dictionary, application schema and application data. Problems will almost always be detected in the lower abstract layers like the application data or application schema, because these layers are used on a daily basis and are understood by more people than the higher layers like the source code of the DBMS. The problem will be detected in either data results after running a query or by detecting a fault in the DBMS structures when attempting to use that structure. Some investigations may be conducted and concluded in the lower layers only. However, when the attacker has attempted to clear his tracks or when the output of the DBMS cannot be explained by the lower levels, the higher levels of the DBMS should be considered.

5.5 FINAL COMMENTS

Because the data model cannot be trusted to deliver correct evidence, the state of the data model should be considered when conducting a forensic investigation on a DBMS. This chapter has systematically presented the methods to transform the data model into a forensically sound state. There are various forensic methods that can be used in various circumstances to conduct a data model sensitive investigation. Depending on the forensic environment, the DBMS investigator should understand the arguments for either a clean or found data model environment, and consequently select a method that is appropriate for the

investigation, keeping in mind to how many of the criteria the method adheres. This chapter has argued that no one method is adequate for every forensic investigation.

5.6 CONCLUSION

This chapter discussed a new way of approaching the data model during a forensic investigation. The data model might be corrupted to deliver false evidence and the forensic investigator should not ignore this fact. The clean and found forensic states were introduced to deal with a compromised data model. Arguments were provided why the found and clean states were necessary. Methods were discussed on how the data model could be converted into a found or clean state. It is important to note that the data model will most definitely not be corrupted in all investigations and most investigations will deliver the correct results without changing the state data model; however, the state of the data model should be considered before an investigation is executed on a DBMS. Chapter 8 includes a section to assist a forensic investigator to decide how the data model should be handled during a forensic investigation. The next chapter will study the data dictionary in a forensic context. Each extensional abstract layer of the DBMS will be considered before the DBMS will be discussed as a whole in chapter 8.

CHAPTER 6 DATABASE DATA

DICTIONARY FORENSICS

6.1 INTRODUCTION

The data dictionary is a specific type of metadata that resides within a DBMS and has to do with the inner workings of the DBMS. There are multiple definitions of the data dictionary in literature [26, 127, 128] and a section in this chapter has been dedicated to provide an explanation of what a data dictionary is. Metadata within a DBMS could be altered to change the inner workings of the DBMS. The data dictionary metadata have the ability to influence the results of a forensic investigation due to the fact that the inner workings of the DBMS have changed. In such an instance the data dictionary should be considered when conducting a forensic investigation. In order to elaborate, we need to consider the data dictionary during a forensic investigation, especially when the DBMS is returning results which indicate a fault in the metadata of the DBMS. The data dictionary is the second in our hierarchy of abstract layers of the DBMS and may therefore influence the application schema and application data as well.

The question that we aim to address in this chapter is how an investigator considers the data dictionary layer during a forensic investigation on a DBMS. This chapter will address this question by firstly explaining what possible alterations or attacks could be executed on the data dictionary layer and thereby illustrate to the reader why this layer is an important consideration during a forensic investigation. Then arguments will be provided why we need to convert the data dictionary into a required forensic state in order to extract evidence from the DBMS. Lastly, various methods will be illustrated on how the data dictionary could be converted into a required forensic state. The data dictionary will then no longer potentially influence evidence extracted from the DBMS.

6.2 WHAT IS THE DATA DICTIONARY OF A DBMS?

Literature proposes a variety of definitions for the data dictionary of a relational database. One study describes the data dictionary in an Oracle context as a central repository of data item information [127]. It continues to explain that the data dictionary is a form of metadata which describes how data are displayed in database reports, provides error

messaging and dictate the use of default data types. Another study describes the data dictionary as tables and related views that enable administrators to see the inner workings and structures of the database [128]. The data dictionary is also described as a mini database management system that manages metadata or a repository of information about a database that documents data elements of a database [26]. Most studies seem to agree that the data dictionary of a DBMS is a type of metadata involved in the inner workings of the DBMS. The definition of the data dictionary becomes important when the application schemas as well as the data dictionary are considered. The application schema of the DBMS is metadata that construct structures such as tables [129]. Consider the following example of what the application schema consists of when making a dump of a table there will be a command as displayed in Figure 6.1; the metadata used to create the table such as ‘Accounts’, ‘UserName’, etc. are considered to be part of the application schema.

```
CREATE TABLE Accounts (ID int(3) auto_increment, UserName  
    varchar(64), Status varchar(10), PRIMARY KEY (ID));
```

Figure 6.1. Simplified example of a create table command containing application schema data

This poses a challenge to distinguish between the data dictionary and application schema. Internal metadata could thus refer to the data dictionary or application schema layer. In this study we would need to make a clear distinction between these two layers of metadata.

Various possibilities were considered on how to solve the problem posed by the ambiguous dividing lines of the data dictionary layer and application schema layer. The two layers could have been combined to form one abstract layer. This was not considered to be an ideal solution, because the internal metadata of the DBMS vary significantly. Metadata could be directly related to an object code level or could be related on a table structure level. The momentous differences in the types of metadata that reside in a DBMS force us to make use of two different abstract layers. The ANSI SPARC model follows a similar approach.

The data dictionary layer was ultimately defined as all metadata of the DBMS that are independent of a particular database in a DBMS. This ensures that the data dictionary metadata are more applicable to the entire DBMS system and not necessarily data which are created by a database administrator. The application schema is defined as all metadata which are applicable to a single database in a DBMS. The application schema is discussed in the next chapter. The dividing lines of the DBMS selected in this study are displayed in Figure 6.2. The data model and application data layers are displayed to provide context.

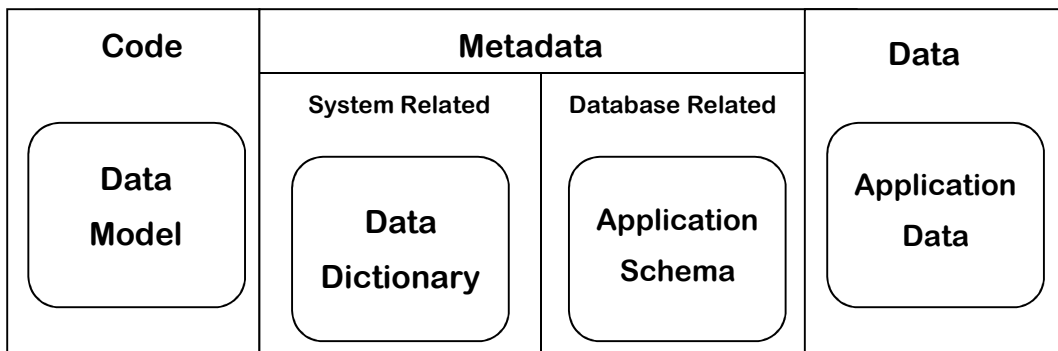


Figure 6.2. Dividing lines between the metadata of the data dictionary and application schema

6.3 DATA DICTIONARY ATTACKS AND ALTERATIONS

This section will explain why it is important to consider that alterations might have been made to the data dictionary when executing a forensic investigation on a DBMS. If alterations are made to the data dictionary, it may not always be obvious to an investigator. Therefore, this section will introduce a variety of possible alterations to the data dictionary which will help a forensic investigator deliberate how a data dictionary can influence forensic evidence or in itself be forensic evidence. The possible alterations introduced in this section are divided into behavioural changes to the data dictionary and alterations to the data dictionary that deliver incorrect results.

The question that springs to mind is why any user wants to attack the data dictionary of the DBMS. First of all, the user who alters the data dictionary is not necessarily an attacker. Changes to the data dictionary might be caused by a user who has made a mistake or has

insignificant knowledge of how the DBMS internal metadata structures work. However, attacking the DBMS on the data dictionary layer is a possibility due to the fact that major changes can be made at this level with sufficient knowledge of the particular DBMS. A reason why someone will alter the data dictionary is to launch an attack on the internal metadata of the DBMS that would not make sense to the average database administrator, thereby ‘hiding’ the alteration to the data dictionary, because it is a layer in the DBMS not understood by the average database user.

Another reason to make alterations to the data dictionary is due to the significant changes that can be made at this level by entering a few SQL commands. Three update SQL commands can change a complete database (as will be illustrated in section 6.3.1). These significant changes can hold various interests to an attacker, such as financial gain, revenge and malice.

6.3.1 Compromised Databases

The first example of a data dictionary structure that can be altered is a metadata structure which stores all the data relevant to databases of the DBMS. Within PostgreSQL DBMS this metadata structure is called `pg_database`. The `pg_database` table is considered as part of the data dictionary layer, because there isn’t a copy of `pg_database` for each database but only one `pg_database` table per cluster which may include several databases. Metadata stored in the `pg_database` table include the database name, owner of the database, encoding methods for each database, a flag that allows or disallows connection to a database, etc.

Several possibilities exist to compromise databases and include the following:

- The database name can be changed and switched with another database name in order to switch an entire data set. Figure 6.3 illustrates how this switching of database names can be achieved. The database name ‘finances’ can be renamed to anything, while another database name ‘temporary’ can be renamed to ‘finances’. Any service referencing the finances database will now have wrong information. This method has been proven in PostgreSQL. Figure 6.3 illustrates that database names can be swapped around in `pg_databases` in order to reference an entire new

dataset. 'finances' is renamed to 'temp4', while 'temporary' is renamed to 'finances'.

- Use the `pg_database` table to compromise the data dictionary layer by disallowing connections to a database to interfere with critical database procedures, such as a monthly backup or report processing.

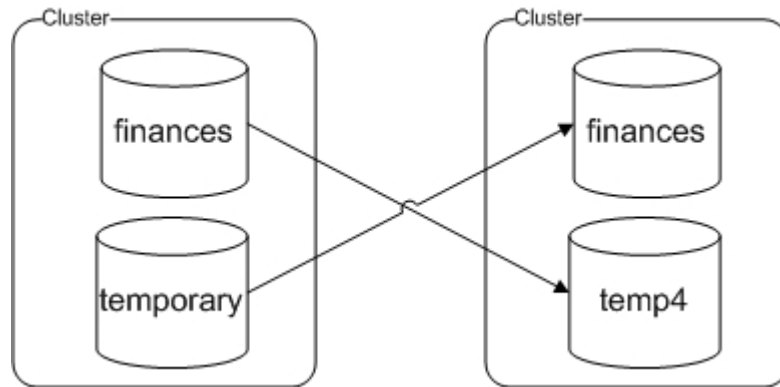


Figure 6.3. An illustration of how metadata can be switched

6.3.2 DBMS User Alterations

In this section we explore data dictionary structures which can be altered in a DBMS. Another example of data dictionary structures which can be altered is the user management tables of the DBMS. User management tables in PostgreSQL include `pg_shadow`, `pg_authid` and `pg_group`, to name but a few. The user management tables are independent from a single database and relevant to all databases in the DBMS; therefore, the user management metadata are considered to be part of the data dictionary.

Possible alterations to the DBMS user management include the following:

- Change a DBMS user's login credentials, such as username and/or password.
- Alter a DBMS user's rights, such as setting a super-user rights flag; disallow or allow a user to create databases; and allow a user to alter the internal metadata tables.
- Alter roles to which a DBMS user belongs.

- Change the group to whom a DBMS user belongs by modifying the DBMS group management metadata structure.

6.4 SELECTING A FORENSIC ENVIRONMENT FOR THE DATA DICTIONARY

In the previous chapter a found environment and a clean environment were introduced. The idea that the internal state of the DBMS should be considered before conducting a forensic investigation was also introduced. The found environment was introduced as a forensic state where no alterations have been made to the metadata or data. An investigation is done on the data “as is”. The found environment allows the DBMS to be mirrored. The clean environment is a forensic state where we have cleaned the DBMS up to a state where the DBMS is not compromised to deliver false evidence. Higher abstract layers of the DBMS are able to influence the lower layers of the DBMS to deliver incorrect or corrupted results. This fact has been illustrated throughout this study. The clean forensic environment is intended to remove the ‘bad influence’ the higher layer of the DBMS may have on a lower layer. It is vital that converting a DBMS to a clean state does not alter the evidence in the DBMS.

The conversion of a DBMS to a clean or found forensic state can be done on an abstract layer level. This means that the data model layer’s state can be changed to clean, while the other abstract layers remain in a found state. In this section we provide arguments why we need to convert the data dictionary layer into a found or clean state.

6.4.1 Arguments for a Clean Data Dictionary

In this section arguments are provided as to why a clean forensic state will be required for the data dictionary layer. These arguments are provided here to enable the reader to comprehend when a clean state should be selected for the data dictionary.

6.4.1.1 Understand Corrupted Results

The first reason why a clean data dictionary layer may be required is to enable the investigator to understand corrupt results. For example, changes to the data dictionary may

make it impossible to do queries on structures within the DBMS. Deleted or modified metadata may corrupt the database in such a way that users cannot access the system, tables cannot be displayed or databases may disappear. A clean data dictionary will correct the action that has corrupted a certain structure that is part of the data dictionary.

6.4.1.2 Eliminate the Data Dictionary Layer

The investigator may need to prove or observe that the data dictionary is not the ‘cause’ of a particular ‘problem’. There may be a mystery that needs to be solved by the forensic investigator but the cause of this mystery cannot be found. In such an instance we need to eliminate ‘innocent’ layers of the DBMS in order to finally determine what layer is the cause of the ‘problem’. The data dictionary layer may be ruled out as a suspect layer if the mystery persists even after the data dictionary layer has been converted to a clean environment.

6.4.1.3 Reveal Hidden Evidence

Another reason why a data dictionary layer should be converted to a clean state is to reveal hidden evidence. For example, a database name change may have been used to switch between the correct data set and a fake data set. The investigator may want to uncover the hidden evidence by making the required changes to the data dictionary in order to reveal the hidden evidence.

6.4.2 Arguments for a Found Data Dictionary

In this section arguments are provided as to why a found forensic state will be required for the data dictionary layer. After reading this section, the user should understand why a found environment may be required for a forensic investigation on the data dictionary layer.

6.4.2.1 Cost of Forensic Investigation

An argument for cost as a reason why a found environment may be required has been provided in chapter 5. The same argument applies to the data model and the data dictionary. Methods used to convert the data dictionary to a clean state often require a lot of effort, expertise and time. On the other hand, to achieve a found state for the data

dictionary layer is often a simple procedure that requires little to no effort. Thus, where cost is a factor and the risk of receiving incorrect evidence from the DBMS is low, the found environment will be a good choice of a forensic environment.

6.4.2.2 Low-Risk Investigation

Another scenario where a found environment will be useful is when it is clear to the investigator that the data dictionary will not influence the results of the investigation. This can be decided after a risk analysis has been done on how probable it is that the data dictionary will influence the evidence queried from the DBMS. If there is no (or a very small) risk that the data dictionary will influence the results of the DBMS, the investigator may do the investigation on the found environment.

6.4.2.3 Test Hypothesis

The investigator may want to test a hunch or suspicion on the DBMS. The data dictionary rarely needs to be cleaned up for this simple procedure. This argument has been provided in chapter 5 and applies to the data dictionary layer as well.

6.5 METHODS TO CONVERT THE DATA DICTIONARY INTO A FORENSIC STATE

In the previous sections we discussed alterations that might warrant a forensic investigation on the data dictionary layer. We discussed potential forensic environments to which the data dictionary could be converted (clean or found). Arguments were provided why the data dictionary might require either a clean or found environment when a forensic investigation was to be executed.

The question that remains is *how* to convert the data dictionary to either a clean or found forensic environment. This section will discuss methods that can be utilised to convert a data dictionary to a clean or found environment.

6.5.1 Methods to Achieve a Clean Environment

In this section methods are discussed to convert the data dictionary to a clean environment state before a forensic investigation is conducted. Remember that a clean environment will

be required if the examiner cannot extract results from the DBMS due to changes to the data dictionary, if the examiner wants to reveal hidden evidence which have been concealed by the data dictionary, etc. The following methods will make it possible to transform the data dictionary into a clean forensic state.

6.5.1.1 Recover the Data Dictionary From Previous Dumps

The first method to convert the data dictionary into a clean state is to recover the data dictionary from a dump that has been made of the entire DBMS before the event of forensic interest has occurred. It is common practice to make regular dumps of a database for backup or development purposes. Therefore, the data dictionary can be recovered to a time when the data dictionary has not been corrupted in order to extract evidence from the DBMS.

The PostgreSQL function `pg_dumpall` is used to dump the data dictionary layer along with the application schema and application data layers. The information in this dump can be used to construct a clean copy of the DBMS. This method will only work if we use a dump that has been made before the event of forensic interest has occurred. The data dictionary is a much more static layer than, for example, the application data layer where data change very frequently. Therefore, an old data dump can prove to be useful to recover a clean copy of the data dictionary.

6.5.1.2 Rebuild the Data Dictionary Manually

Consider a situation where the data dictionary has been renamed to make use of the wrong database and we cannot make use of the renamed database because there are too many external and internal systems depending on the correct database name. We may want to test with precision how the system is supposed to work. The only way to do that is to change the data dictionary to reference the correct database. Therefore, we need to alter the data dictionary and cannot work on the data dictionary as it was found. The database will then be renamed back to what it is supposed to be and the test can be run.

This is an oversimplification of how the data dictionary can be cleaned up manually. In more complex circumstances it may be required to do this cleansing on a larger scale. The storage locations of tables of a database may be modified in such a way that the table is inaccessible (see the `pg_tablespace` metadata table of PostgreSQL for more information). The table is effectively deleted if we cannot reference it with the DBMS.

A simple reset of the `pg_tablespace` to its default values will fix the problem; however, the investigator needs to search for this inconsistency in the data dictionary metadata. A manual search through the data dictionary in order to find inconsistencies is another method of converting the data dictionary to a clean forensic environment.

6.5.2 Methods to Achieve a Found Environment

This section will discuss examples of how to transform a data dictionary into a found forensic environment. Recall that the found forensic environment will be used when the data dictionary should be examined as it was found. There are some options of how to achieve a found forensic environment for the data dictionary.

6.5.2.1 Live Analysis

The first method to achieve a found environment is to execute an investigation on the data dictionary as it was found. It may sound like an odd notion to achieve a found environment by applying no effort to the data dictionary at all. However, this live analysis is a state of the forensic environment which we suspect will be used in most forensic investigations. This live analysis of the data dictionary is categorised as an investigation in a found environment.

6.5.2.2 Mirror of the Data Dictionary

Another method of transforming the data dictionary into a found environment is to mirror the entire DBMS onto another host. There is no easy way to mirror the data dictionary in isolation. A mirror can be done by mirroring the hard drives which contain the installation of the database. The term “hard drives” are used in plural due to the fact that a database may exist on more than one hard drive, especially on servers.

6.5.2.3 Backup Server

The final method to achieve a found forensic environment is to make use of the data dictionary as it was found on a backup server. This technique has been discussed in chapter 5 and also applies to the data dictionary found environment. If a data dictionary is corrupted on a database server, there is a chance that the same corruption has not taken place on the backup server. The probability of using this method successfully is lower than in the case of the data model discussed in the previous chapter. The reason for this is that SQL commands are mostly used to alter and corrupt the data dictionary. These SQL commands are copied over to the backup server and executed. No SQL commands are executed to alter the data model and therefore the alteration will not be executed on the backup server. It remains a possibility that the backup server may not have been compromised and it is an option to achieve a found forensic environment.

6.6 CONCLUSION

This chapter focused on the data dictionary abstract layer and how the data dictionary should be considered in a forensic investigation. It is important to note that although there have been similarities between the data model and data dictionary, each abstract layer presents its own unique challenges in a forensic environment. Ultimately we aim to find the correct forensic environment for each layer of the DBMS before conducting a forensic investigation to ensure that evidence is trustworthy. The application schema is the next abstract layer in the hierarchy and will be discussed in the following chapter.

CHAPTER 7 DATABASE APPLICATION

SCHEMA FORENSICS

7.1 INTRODUCTION

The application schema describes the physical structure of the data [112]. A single database can hold multiple application schema objects, such as tables, views, triggers, procedures, indexes, keys and other database designer objects [113]. 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 a forensic investigation is required on the application schema level, the application schema should be prepared to create an environment where the application schema can be investigated. A 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 [101]. 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 chapter will answer this question by firstly pointing out why a forensic investigation on the application schema level can be required. Several scenarios will illustrate how the application layer can be damaged or altered to point out why a forensic investigation can be required on the application schema layer.

Thereafter, the clean and found forensic environments will be discussed in an application schema context. 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 on how a forensic investigation environment on the application schema of the DBMS can be prepared. 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 as evidence in a court of law.

7.2 ARGUMENT FOR APPLICATION SCHEMA FORENSICS

This section will discuss some of the scenarios where modifications may occur on the application schema layer and consequently warrant a forensic investigation. 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 has happened on the DBMS. It is important to understand that whenever this chapter mentions application schema forensics, it refers to a forensic investigation that focuses on the application schema layer in particular. The DBMS consists of four abstract layers, but this chapter will investigate the forensic possibilities on the application schema layer of the DBMS.

There are various reasons why a user may want to transform the database application schema in a way that enables the application schema to be investigated. Customer loyalty programs, targeted marketing, customer services and the management of practically every corporate resource depend on databases. There are endless reasons why a user may intentionally change, damage or delete application schema structures [114]. Users may also unintentionally alter or damage the application schema and a forensic investigation will be required to determine what has 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 can be the perfect layer for attacks where the attacker can change data in such a way that he hides the evidence to a certain extent. Alternatively, an attacker may choose to alter or damage the application layer, because it is the only portion of the database to which the attacker has access. A forensic investigation will be required to determine what has happened on the application schema if an attack of this nature occurs.

Application schema alterations are usually done to develop and maintain the database but may be used to execute criminal, unethical or damaging commands. The possible alterations to the application schema are divided into changing the behaviour of the application schema, altering the application schema to deliver incorrect results and damaging the application schema. The scenarios discussed in this section are by no means an exhaustive list of possible alterations to the application schema that may warrant a forensic investigation. There are too many possibilities to mention and a list of such a nature will become outdated and incomplete, based on the rapid evolution of databases. In the following sections the comprehension of the different categories of alterations to the application schema is of utmost importance. This will be illustrated by making use of some scenario examples.

7.2.1 Damaging the Application Schema

The first category of application schema alterations that may warrant a forensic investigation is damages to the application schema. The scenarios mentioned in this section relate to application schema alterations that cause damage to the application schema where the application schema structures need to be recovered in order make use of those structures again. The scenarios include damage to the schema with SQL drop command, altering access privileges and corrupting tables.

7.2.1.1 Damage Schema with the SQL Drop Command

A database user may 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 unsatisfied customers. Even if

backups of the database were kept, it might take some time to rebuild the table. Furthermore, 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.

7.2.1.2 Alter Access Privileges

Access privileges are granted and revoked at regular intervals on databases [115]. 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;
```

Figure 7.1. An SQL command to remove access privileges from a certain table for a certain user

When a Web server loses connectivity to a table on which it depends 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.

7.2.1.3 Corrupt Table

Another manner in which the application schema can 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 describe the data warehouse itself [116]. The metadata include the description of the data model, database design definitions, descriptions of transformations and aggregations, etc. Some of this metadata are directly linked to the application schema of the database. For instance, a database table is defined by making use of metadata. Furthermore, aggregation functions

are mapped in metadata and directly influence the way in which queries are done on the application schema. This metadata can be altered to change results of queries.

A PostgreSQL DBMS has been used to test how a table can be corrupted via metadata. The metadata which construct a table reside 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. There are various possibilities of 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 may be required to determine what has happened, who is responsible and what the goal of the alteration is (to damage the data, an honest mistake, for personal gain, etc.).

7.2.2 Application Schema Alterations to Deliver Wrong Results

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

7.2.2.1 Column Swop

The first application schema alteration that can influence the DBMS to deliver erroneous results is to swop a column in the metadata. The authors of this paper have compromised the application schema by swopping two column names within a table [102]. In PostgreSQL the `attnum` sequence in the `pg_attribute` table has been 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 7.3 illustrates what commands can be used to swop two columns in the PostgreSQL DBMS. The `pg_attribute` table consists of metadata which construct the tables of the PostgreSQL DBMS. If this metadata are changed, the application schema is altered

significantly. A forensic investigation may 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 7.2. Commands to swop two columns of a table in the application schema

7.2.2.2 Operator Swop

Another alteration of the application schema metadata that may warrant a forensic investigation is an operator swop. This alteration will influence the DBMS to deliver incorrect results on a query. The DBMS makes use of metadata to define what processes should be used to do mathematic operator calculations. The metadata will link the plus operator (+) to summing processes. This metadata that link operators with processes can be altered to link an operator to the wrong process. For example, the division operator may be changed to make use of multiplication processes. Every time the division operator is used, a multiplication will be carried out. This is defined as an operator swop. Figure 7.4 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 7.3. Commands used to alter the division operator to multiply

7.2.2.3 Create View to Replace Table

Creating a view to impersonate a table to hide data is another way in which the application schema can be altered to deliver the wrong results. Imagine a situation where a table with the name *MinimumCost* exists. The table is mostly used on which to do *SELECTs* and is rarely updated. The table is then renamed to *MinimumCost2* and a view is created with the name *MinimumCost* to replace the table. However, the view can be set up to hide some rows from the original table. The following SQL command can be used to create the view to hide some rows from the initial table:

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

Figure 7.4. An SQL command which may enable a view to replace a table

7.2.2.4 Aggregation Damage

DBMSs use aggregation functions in SQL commands to calculate and summarise data [4]. These aggregation functions are captured in the metadata in a similar way than the mathematic operators mentioned above. The *pg_aggregate* table in PostgreSQL holds the metadata about how to handle aggregation functions [8]. 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 7.5 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.

```
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 7.5. Commands to alter an existing aggregation function

7.2.3 Database Behaviour Alterations

The following section will illustrate why a forensic investigation may be required due to alterations to the application schema which change the behaviour of the database. The examples discussed below include slowing the DBMS down by dropping tables and modifying the storage engines of a table.

7.2.3.1 Slow the DBMS Down by Dropping Indexes

The first way to change the application schema in order to affect the behavior 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 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 has happened.

7.2.3.2 Blackhole Storage Engine

Another method which may warrant a forensic investigation through an alteration in the application schema is the use of different storage engines. MySQL supports various storage engines which act as handlers of different table types [117]. 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 is a large number of database slaves plus a busy master machine [118]. Network load can become significant even when compression methods are used. The blackhole storage engine acts as a blackhole that accepts data but throws it away and does not store 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 may enable a malicious user to prevent a database from updating information. This example also demonstrates that some features in the DBMS may be used to exploit the application schema. The current study will seek to find a solution for all DBMSs.

7.2.3.3 Custom Storage Engines

Furthermore, database experts can write their own storage engines to perform potentially malicious tasks. This 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 how to write a custom storage engine can be found in [119].

7.3 SELECTING AN APPLICATION SCHEMA FORENSIC ENVIRONMENT

The previous section has illustrated that the application schema layer can be altered to deliver incorrect evidence. Therefore, it also needs to be considered that this layer can deliver corrupt results when conducting a forensic investigation on a DBMS. This section will focus on how to approach an application schema forensic investigation. Environments will be introduced in which a forensic investigation can be conducted and several advantages of the various environments will be mentioned. A forensic environment should be selected that will best fit the forensic scene.

Two possible forensic environments can be prepared for evidence collection based on the forensic scenario. These environments are the found and clean forensic environments which have been introduced in previous chapters. The found forensic environment is a forensic environment where the forensic investigation takes place on the application schema as it was found. Mirrors of the complete DBMS are allowed in this forensic environment. The clean environment represents a forensic environment where the application schema or parts of the application schema have been cleaned up in order to reveal evidence that will otherwise not be visible or apparent. Arguments have been provided for the data model and data dictionary but the following sections will discuss why it may be necessary to have either a found or clean environment for a forensic investigation in the application schema layer in particular.

7.3.1 Arguments for a Clean Environment

The clean environment is a setting where we have ensured that an alteration will not change the output of the DBMS. It is important to remember that a clean state differs from a post-mortem state that is commonly known in traditional digital forensics. A clean state is not a copy of the evidence that needs to be analysed but rather a copy of the evidence program that runs like the original copy and from which we can query results. This means that the clean environment is set up to run like the original DBMS but we are sure that an alteration to the application schema is not corrupting the output that we receive from the DBMS. A clean environment poses several advantages to application schema forensics which are discussed in the following sections.

7.3.1.1 Application Schema Structure Recovery

The first argument for a clean forensic environment where parts of the application schema may be cleaned is the repair of application schema structures. If an application schema structure (like a table) has become damaged on the original DBMS, it may be necessary to recreate that application schema structure on a new DBMS in order to display the content of the application schema structure. It is risky to attempt to repair application schema structures on the live environment since evidence may be altered. A clean environment will minimise the risk of damaging evidence when repairing application schema structures.

7.3.1.2 Clean Metadata

The second argument for a clean application schema forensic environment is that clean metadata may yield forensic results. Clean metadata refer to setting application schema metadata structures to their default settings. In some instances the metadata of the DBMS may have become corrupted or may have been altered, causing results to be investigated. Clean metadata may reveal application schema structures that may not have been visible to the average database administrator. Clean metadata may also restore basic DBMS functions in order to search for forensic evidence. Caution should be taken about which metadata are used in a clean state and which metadata are retained from the found DBMS. 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.

7.3.1.3 Integrity

A DBMS can be modified in various ways to deliver 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 to not deliver results. A clean environment provides a degree of integrity where the investigator can be sure that evidence results are not influenced by modifications done to the application schema.

7.3.1.4 Elimination of Layers

Imagine a situation where the database is delivering 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 but in most cases the various layers of the DBMS will need to be eliminated until the cause of the problem is identified. This elimination can be done by cleaning up a particular layer of the DBMS to observe if the unpredictable results persist. A clean environment may therefore assist in eliminating layers of the DBMS to find the source of a problem.

7.3.2 Arguments for a Found Environment

A found environment is a state of the DBMS where the DBMS is the same as when an event of forensic interest has occurred. The found environment may also refer to a forensic environment where the same machine, where the DBMS has been installed originally, is not used but the DBMS has been mirrored onto another machine. It is important to remember 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 arguments for a found application schema environment are discussed in the following sections.

7.3.2.1 Cost of Forensic Investigation

The first argument for a found forensic environment where the evidence is analysed as it has been found, is the cost of the investigation.

There are a couple of reasons why a forensic investigation on a found environment is more 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.

We expect that the cost difference between the found and clean environment in the application schema layer will be much less than the data model layer, because the complexity required to set up a clean environment in the data model layer is significant.

7.3.2.2 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. This applies to all abstract layers discussed thus far.

7.4 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, as has been done in the two previous chapters. This section will be split up into a discussion of methods to achieve a clean environment and methods to achieve a found environment for an application schema forensic investigation. Several methods will be provided for the clean and found environments. Finally, we briefly discuss some known evidence searching techniques.

7.4.1 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.

7.4.1.1 Rebuild from Previous Dumps

This method applies to the application schema as the data dictionary but there are some differences. It is a method to achieve a clean application schema environment by rebuilding 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 has occurred. Dump files include an application data section which is generally accepted 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 affects the application schema will 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. A decision on the state of the data model should be made by referring to chapter 5. However, our focus is on the application schema and 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.

7.4.1.2 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 model, database design software that has not been synchronised with the application schema of forensic interest (such as MySQL Workbench) or official drawn-up sketches [120]. The application can be rebuilt by running SQL commands that will 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 take a lot of time when using database design software which can build the application schema automatically.

7.4.1.3 Rebuild from Pieces

Another method to achieve a clean application schema forensic environment 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 may be rebuilt from the leftover data to some extent. This leftover data may 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 has been destroyed. This method requires a lot of expertise and effort but may be the only forensic option in some cases.

7.4.1.4 New Metadata Template and Metadata Alterations

The last method to achieve a clean application schema forensic environment 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 can 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.

7.4.2 Found Application Schema Environment

The found application schema environment is a setting where the application schema can be examined forensically as it has been 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 following sections will discuss methods to achieve a found application schema forensic environment.

7.4.2.1 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 has 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.

7.4.2.2 Copy of the Application Schema

Another way in which 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 has 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.

7.5 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 in the context of the application schema layer.

In many cases the DBMS itself will be used as a forensic tool to deliver evidence, because only the DBMS itself can interpret its own encrypted data and structures where most of the evidence related to the DBMS will be stored. Therefore, we have taken several steps to

ensure that the extensional layers can be trusted. The previous three chapters have considered the each extensional layer of the DBMS in a forensic context. The chapters explain how an abstract layer can be compromised and influence other extensional layers or the actual data received from the DBMS. Each extensional layer has been illustrated in a forensic context to assist the investigator to combine the correct forensic state of each layer in order to conduct an investigation on an entire DBMS. The discussion of each extensional layer has also equipped the investigator with an in-depth understanding of the role of each layer and to what extent a layer can influence the forensic investigation.

The following chapter will combine the layers of the DBMS as a whole to enable a forensic investigator to execute an investigation on a compromised DBMS while ensuring that the evidence can be trusted.

It might seem odd that a chapter for the application data layer is not included at this stage. The reason for this is that the application data layer does not influence any other layers in a compromised DBMS. The results of the application data layer will mostly be used as evidence output and the goal is to ensure that the extensional layers of a compromised DBMS are not influencing one another or the application data.

CHAPTER 8 PRACTICAL APPROACH AND RESULTS

8.1 INTRODUCTION

The previous chapters discussed three layers of the DBMS in isolation. Each layer has been discussed in a forensic context and how a forensic investigator can deal with that DBMS abstract layer.

In this section, an approach will be discussed of how all layers of the DBMS can be viewed collectively in a forensic context. An investigator who is required to do a forensic investigation on a DBMS will have to consider all layers of the DBMS when conducting a forensic investigation. This does not mean that the individual abstract layers are not relevant anymore; in fact, the knowledge gained from the previous chapters will assist throughout the forensic investigation.

The respective forensic solutions discussed in the previous chapters (data model, data dictionary and application schema) will be joined to illustrate how the methods of this study could be put to practice. However, to view the entire DBMS in a forensic context will require a slight extension of the approach discussed in previous chapters. The combination of the abstract layers is investigated in this chapter and the abstract layers can only be viewed individually after the combination of abstract layers has been considered. This chapter will demonstrate how the methods and approaches discussed in previous chapters could be applied to the whole DBMS by discussing the extended approach required for an investigation on a whole DBMS and illustrating a practical example of how a forensic investigation could be conducted on a DBMS. The complete DBMS investigation approach and example are united into one chapter at this stage to retain a logical flow of information.

The approach required to do a forensic investigation on the whole DBMS is discussed in section 2 because it varies slightly from the forensic approach applied to the individual abstract layers. A DBMS forensic matrix will be introduced to assist an investigator in

understanding the possible states in which the DBMS could be during an investigation and in what scenarios the DBMS might find itself in a particular state. After laying out all possible states the DBMS could be in during a forensic investigation we need to select one of the states for a forensic investigation. Risk profiling will be discussed to select which state will be the most effective for acquiring accurate evidence from the DBMS. Thereafter the complete evidence identification process for the whole DBMS will be discussed so that a forensic investigator could prepare the DBMS to extract trustworthy information from it during a forensic investigation.

Section 3 will illustrate a practical example of how a forensic investigation could be carried out on the whole DBMS. The practical example is included here to test our theories in practice, since theory might often differ from practice and the authors aim to prove that our theories could work in practice.

Another reason why the practical example is presented is because if forensic investigators need to apply our theories in practice when doing a forensic investigation on a DBMS it should be simple to make the mind-shift from theoretical discussion to practical implementation. A DBMS will firstly be compromised to obtain an actual test dummy of a compromised DBMS. The process of evidence identification will then be followed to investigate the DBMS. The approaches and methods discussed in this chapter as well as the methods and approaches discussed in previous chapters will be used to investigate the DBMS. The results of our practical example will be explained. Finally, section 4 will discuss our findings in this chapter.

8.2 APPROACH

In this section we will describe the theory of approaching a forensic investigation on an entire DBMS. A forensic investigation on a DBMS will rarely be conducted on a single abstract layer only. The DBMS should be viewed as a single entity to be investigated that consists of various abstract layers. The abstract layers and forensic methods discussed in the previous chapters will be combined in this chapter to enable the reader to see the entire DBMS in a forensic context.

8.2.1 The DBMS Forensic Matrix

This study introduced the clean and found environment of the metadata abstract layers of the DBMS. Each of the data model, data dictionary and application schema layers of the DBMS will either be in a clean or found environment when conducting a forensic examination. The states of the metadata abstract layers can be combined into several groupings of which one grouping can be selected during a forensic investigation. One of these groupings will be called ‘a scenario’.

Table 8.1 illustrates the DBMS forensic matrix which suggests that there are eight different scenarios for conducting a forensic investigation. Here a scenario refers to the combination of the abstract layer states to be achieved in a DBMS under investigation. The aim of the DBMS forensic matrix is to assist the investigator in assuring that the correct scenario is selected to acquire accurate evidence from the DBMS at hand.

Another benefit of the DBMS forensic matrix is to lay down all the possible combinations of abstract layer states that are possible within the DBMS and in so doing, make the forensic approach presented in this study more understandable and manageable. For example, in the first scenario of Table 8.1 (scenario ID 1) a forensic investigation will be executed where the data model, data dictionary and application schema are in a found state. In scenario 7 the data model and data dictionary are converted to a clean state and the application schema is in a found state.

Note that the application data layer is not converted to either a found or a clean environment. The reason for this is that there is no need to convert the raw data to another state. The raw data of the application data layer cannot influence the other layers of the DBMS in any way and therefore do not need to be converted to another state during a forensic examination. We are able to trust the results of the application data layer without converting the application data into another state. Therefore, the application data is denoted with an x in Table 8.1. This does not mean that a data record cannot be changed by an attacker to deliver incorrect data. It means that we are able to acquire application data

records from the DBMS and know that they truly reflect the state of the record in the DBMS.

Consider the following example: The application data layer does not need to be converted to a clean or found state, because if an attacker has doubled a payment amount in the application data layer we can make use of the logs and directly query the DBMS to prove that the amount has been doubled. On the other hand, the data model layer may be manipulated to show that the payment amount is not doubled but for an undeclared reason a doubled amount is still paid out. A clean data model environment is required to stop the data model from doubling the amount, while a found data model is required to view the malicious code entered into the DBMS. This example illustrates that a clean and a found environment are crucial options when dealing with the data model layer but are not required for the application data layer.

Table 8.1: The Database Management System Forensic Matrix*

Scenario ID	Data Model	Data Dictionary	Application Schema	Application Data
1	<i>F</i>	<i>F</i>	<i>F</i>	<i>x</i>
2	<i>F</i>	<i>F</i>	<i>C</i>	<i>x</i>
3	<i>F</i>	<i>C</i>	<i>F</i>	<i>x</i>
4	<i>F</i>	<i>C</i>	<i>C</i>	<i>x</i>
5	<i>C</i>	<i>F</i>	<i>F</i>	<i>x</i>
6	<i>C</i>	<i>F</i>	<i>C</i>	<i>x</i>
7	<i>C</i>	<i>C</i>	<i>F</i>	<i>x</i>
8	<i>C</i>	<i>C</i>	<i>C</i>	<i>x</i>

* *F* represents a found environment, *C* represents a clean environment and *x* represents no environment application.

In the remainder of this section it will be discussed why some of the scenarios mentioned above are relevant and important. It is expected that not all scenarios are of equal importance. For example, a complete found investigation in scenario 1 is expected to be used most of all scenarios, because this scenario requires little to no alteration to the states of the abstract layers and the authors of this study certainly do not expect every DBMS to be compromised.

On the other hand, scenario 6 might be used rarely in a DBMS forensic investigation, because there are few situations where the data model has been manipulated along with the application schema but the data dictionary has not been altered. We suspect that not all scenarios will be equally popular.

Consider the first scenario which is the popular purely found environment (FFF). This scenario will be used when the forensic investigator decides that the higher layers of the DBMS are not influencing the evidence and therefore no layers have to be cleaned up. An example of such a scenario is when the forensic investigator can determine that the higher layers of the DBMS have not been altered to commit the act under investigation. This may be when the DBMS is only a witness or it is not allowed to achieve a clean state due to company policies. On the complete other side of the scale is the purely clean environment (CCC) where the investigator decides to clean up three layers of the DBMS. This scenario will typically be used when a DBMS has been corrupted or destroyed to such an extent that we have to clean up the entire DBMS in order to extract evidence from it.

However, it is not always necessary to clean up the all the layers of the DBMS if a compromise is suspected on the DBMS. Sometimes only a single abstract layer needs to be cleaned up and the remaining abstract layers remain in a found state. Scenario 2 (FFC) will be used when the application schema of the DBMS has been altered to such an extent that we have to clean up the application schema in order to obtain forensic evidence from the DBMS. An example of a problem that may require scenario 2 is where the sum operator of the DBMS has been corrupted to multiply (instead of summing values) and we need to see

what the original results are. In this way the forensic investigator may find evidence that has been hidden by corrupting the sum operator.

In scenario 3 (FCF) the data dictionary layer of the DBMS is compromised and needs to be cleaned up in order to acquire results from the DBMS. In chapter 6 we illustrated how a database could be changed completely by switching the database with a false copy. Here the investigator may need to clean up the data dictionary to see what data should have been used.

Scenario 5 (CFF) requires the data model of the DBMS to be cleaned up in order to conduct a forensic investigation on the DBMS. In this scenario a hacker may have changed the code of the DBMS in such a way that results from the DBMS are corrupted by the code of the DBMS, results from the DBMS are displayed incorrectly, logging off the DBMS is reporting false events or not reporting events at all etc.

Scenarios remain that represent a combination of clean and found states where the investigator determines that more than one abstract layer needs to be converted to a clean state. Scenario 4 (FCC) represents a scenario where all the metadata of the DBMS are not trusted and need to be cleaned up. This clean-up could be done to reveal evidence not possible to obtain in a found environment or to test if the DBMS behaves differently if the metadata are cleaned up.

There will seldom be a situation where a forensic investigator uses scenario 7 (CCF), and decides to clean up the data model and the data dictionary only without cleaning up the application schema as well. However, this scenario may be used if the forensic investigator decides to clean up the layers one by one and solves the investigation before cleaning up the application schema layer.

8.2.2 Risk Profiling

In the previous section eight possible scenarios were presented that might be used during a forensic investigation. How do we select what scenario to use for a forensic investigation?

The next useful extension of our approach to conduct a forensic investigation on the whole DBMS is to do risk profiling on the layers of the DBMS to select an appropriate scenario from the DBMS forensic matrix. The types of forensic investigations that can be required vary significantly in nature. A list of situations that links to one of the eight scenarios of a DBMS forensic investigation will barely ever be complete and would be outdated very quickly. Therefore, we need an approach that will work for a variety of situations.

The forensic investigator is required to do risk analysis before conducting a forensic investigation on a DBMS. We need to determine what scenario we are going to make use of by finding the scenario that minimises the risk of extracting wrong evidence from the DBMS. Figure 8.1 illustrates a risk matrix that can assist the forensic investigator in minimising the risk of delivering false evidence. This risk analysis has to be done for each layer individually. The probability axis in this figure relates to the probability that the layer has been compromised. The consequence axis relates to how severe a compromise may affect the evidence.

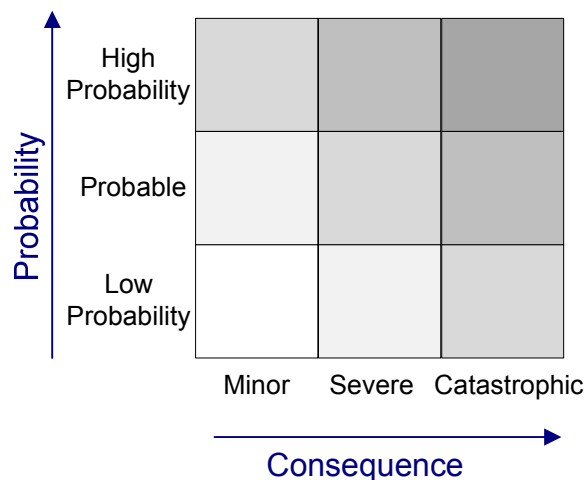


Figure 8.1. A Risk Matrix

If an abstract layer of the DBMS falls into the *low probability-minor* quadrant then the investigator should use a found environment for that abstract layer. The reason for this is that there is a low probability that the abstract layer has been compromised and a compromise in this instance will have a minor effect on the evidence. A *high probability-*

catastrophic quadrant result for an abstract layer will mean that the forensic investigator will have to make use of a clean environment for that abstract layer. This is because there is a high probability that the layer has been compromised and the compromise has a catastrophic effect on the evidence. Finally, if the risk of the layer falls somewhere in the middle of the risk matrix (e.g. *probable-severe* quadrant, *high probability-minor* quadrant) then the decision is up to the forensic specialist to either use a clean or a found environment for the specific abstract layer.

If a forensic investigator has to do an analysis on a DBMS that has been attacked by a rootkit and the rootkit is still running on the machine, the risk profile of the data model layer will most likely fall in the *high probability-severe* quadrant or the *high probability-catastrophic* quadrant. This risk profile will then require the investigator to make use of a clean data model environment.

In another example, if a financial employee with a limited computer background changes a value in a finances table to commit corruption, the data model layer will likely be in the *low probability-minor* quadrant. This is because there is a very small chance that this employee has changed the code of the DBMS. The investigator has to remain mindful that there may be a compromise that has not been expected.

8.2.3 Evidence Identification Process for DBMS

The discussion of the DBMS forensic matrix, risk profiling of abstract layers and the individual layers viewed in a forensic context (in previous chapters) has prepared the reader to view the holistic process of identifying forensic evidence during a forensic investigation on a DBMS. In this section a process will be illustrated of the forensic identification process on the entire DBMS. Figure 8.2 displays an evidence identification process for a DBMS.

The remainder of this section is used to explain the process in Figure 8.2. Once a forensic investigation is required on a DBMS, a forensic investigator will receive a request for a forensic investigation and a forensic query. A forensic query is typically something like:

“Determine if the passwords were stolen off our database”. This is part of the *investigation required on DBMS* process. The forensic investigator then has to *determine the risk profile of each layer*. This relates to the risk matrix that has been discussed in the previous section. The risk profile has to be done for the data model layer, data dictionary layer and application schema layer. The risk profile of each abstract layer will recommend either a clean or a found environment for the abstract layer.

In the *Select Forensic Scenario* phase the forensic investigator selects which scenario of the DBMS forensic matrix will be used, based on the risk profile of each abstract layer. At this stage the forensic investigator may have decided to select scenario 3 which is the FCF scenario. Here the investigator has determined that the data dictionary is a “high risk” and needs to be in a clean state.

After the forensic investigator has selected a forensic scenario from the DBMS forensic matrix, methods need to be selected on how the layers of the DBMS will be converted to the correct states. These methods have been discussed in chapter 5 to 7 for the data model, data dictionary and application schema layers respectively. Here it is important that the forensic investigator selects methods for each abstract layer that are compatible with each other. The investigator then needs to *implement the methods* that have been selected in order to convert the abstract layers to the correct state.

Now the DBMS is in the required state and can be searched for evidence. This can be done by either querying the DBMS, viewing log files etc. If forensic evidence is found, the evidence identification process ends but if no evidence is found, we may need to change the scenario in order to find evidence.

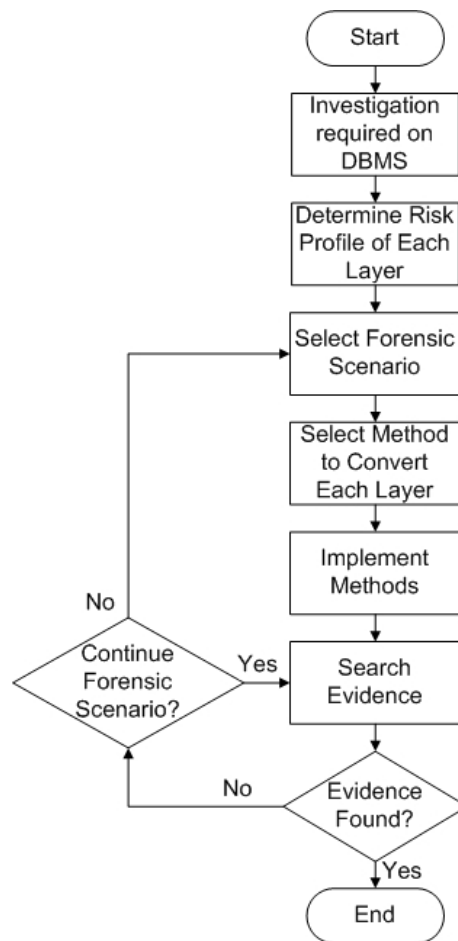


Figure 8.2. A DBMS Forensic Evidence Identification Process

8.3 PRACTICAL EXAMPLE

The remainder of this chapter will illustrate how selected methods from this paper can be used to achieve a forensic environment that is suitable for a forensic investigation and which will enable us to acquire trusted evidence. It will be illustrated how the DBMS can be compromised in order to obtain a real-life compromised DBMS on which to test our methods. It will be illustrated how the DBMS can be transformed into the desired state for a forensic investigation. The examples of this chapter have been published by the authors [102]. This section will specifically focus on the evidence identification process of the forensic process. The evidence identification process not only allows one to locate the key evidence, but also maintains the integrity and reliability of that evidence [95].

In order to conduct the experiments of the collection scenarios successfully three virtual Ubuntu machines have been set up with a psql installation each. The one Ubuntu machine is used as the machine which always heralds the potentially compromised DBMS installation. This machine requires at least a database, tables and records in order to have some application schema and application data to work with. The use of the third machine is optional, depending on the scenario chosen. For a simple mirroring of the DBMS onto another machine, only the first and second virtual machines are required. As the scenario becomes more complex than that, the third machine will become essential.

The Ubuntu operating system is used in this study due to its popularity and ease of use. The PostgreSQL DBMS is used because it is an open source DBMS where the source code of the DBMS is available and it is easier to compromise the DBMS for our testing purposes. PostgreSQL is also a complete DBMS system that consists of enough data model, data dictionary, application schema and application data structures to utilise for purposes of this study.

8.3.1 Compromising the DBMS and Setting Up a Test Environment

To illustrate how a DBMS can be compromised holds two benefits at this stage. The first is to illustrate practically how some of the layers of the DBMS can be compromised. Up until this point compromises to the DBMS have mostly been discussed in theory.

The second benefit of illustrating how a DBMS can be compromised is to prepare a compromised DBMS on which we can apply our forensic process and methods.

In order to compromise the DBMS on one of the virtual machines a small database was created and populated. Then changes were made to the levels of the DBMS to represent compromises to that layer of the database. The commands in Figure 8.3 were used to configure the DBMS which would be compromised. Figure 8.3 illustrates that a database has been created with some tables, a schema, a trigger and a function.

- `su - postgres`
- `/usr/local/pgsql/bin/createdb test`
- `/usr/local/pgsql/bin/psql test`
- `create table schema (name varchar(20),number int,highnumber int);`
- `create table data (id varchar(5),name varchar(20),salary float,CONSTRAINT id_con PRIMARY KEY(id));`
- `insert into data values ('432','RandomNames',1500);` - *repeated with different random values*
- `insert into schema values ('RandomNames',21,100);` - *repeated with different random values*
- `create view dataview as select id,salary from data;`
- `create unique index id_idx on data (id);`
- **Function:** `create function increase() returns trigger as $$
begin
update salaries set salary = x where surname = 'Y';
end
$$ language plpgsql;`
- **Trigger:** `create trigger increase_trigger
after update on salaries
for each row execute procedure increase(surname);`

Figure 8.3. Configuration of Compromised DBMS

After the basic database structure had been built, the DBMS was compromised. The data model and the application schema were compromised in order for tests to be conducted on a real compromised database. The data model was compromised by changing the welcome message within the source code of psql and reinstalling psql. This was done by exchanging the welcome phrase with “Compromised welcome..” in the `/src/bin/pgsql/psql` file within the folder containing the psql source code.

This means that each time a user logs into a DBMS with the compromised data model, it can easily be identified by the welcome message. The altered welcome message makes it easy to identify when we are dealing with a compromised data model and when the data

model has been cleaned up to such an extent that the correct welcome message is recovered.

- `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 8.4. Commands used to compromise application schema

The application schema was compromised by swapping two column names within a table. 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. Refer to chapter 7 on more examples of how the application schema could be compromised. The compromised data model and application schema will help us to easily identify whether a compromised layer is present or not. The code in Figure 8.4 has been used to compromise the application schema. The result is a compromised DBMS of which the data model layer and the application schema layer are compromised.

8.3.2 Risk Profiling

After the test environment has been created, the evidence identification process is followed. The first step of following the evidence identification process discussed earlier in this chapter (displayed in Figure 8.2) is to do risk profiling on the compromised DBMS. After the DBMS has been compromised and odd behaviour detected on the DBMS, an investigator will be notified that a forensic investigation is required on the DBMS. At this stage the forensic investigator has no idea that the DBMS has been compromised. However, the odd DBMS behaviour that has been reported – like results extracted from the wrong table – will assist the forensic investigator in the risk profiling of the layers. The

first task of the forensic investigator is to do this risk profiling on the various layers of the DBMS in order to decide if there is a risk that the relevant layer has been compromised. He has to determine what the consequences will be for querying evidence dependent on that compromised layer.

The ‘welcome’ message that is compromised in the data model layer will most likely force the investigator to select a clean data model environment. The investigator may be skilled enough to realise that the swapped columns are most likely caused by the application schema layer of a psql DBMS and therefore he will select a clean environment for the application schema layer. A found environment will be selected for the remaining data dictionary layer because we do not foresee that the data dictionary is compromised.

Note that we have made the correct decision here quite easily. In reality, the risk profiling is a tool that provides the investigator with the best guess. Ultimately, the selection of the correct environments may be an iterative process where the correct scenario is only selected after a couple of attempts. This iterative process is worked into the evidence identification process discussed earlier in this chapter.

8.3.3 Select Forensic Scenario

The next step of the evidence identification process is to determine which scenario to use. A structured investigation environment will be built by making use of the various layers of metadata and data from the DBMS. The evidence identification process suggested earlier in this chapter is designed to be re-usable for various types of DBMSs. The forensic matrix proposed is a binary matrix that ranges from CCC to FFF. Each of the three abstract layers of the DBMS (data model, data dictionary and application schema) can either represent a “C” or an “F” in this matrix.

To illustrate the iterative process, in our example the investigator has first selected a CFF environment by only profiling the data model layer as a risk. Thereafter the forensic investigator has profiled the application schema as a risk as well and the CFC scenario is used to solve the forensic investigation. Several scenarios will be discussed along with our

example in the remainder of this chapter in order to give the reader an understanding of more than one scenario.

8.3.4 Method to Convert Environment

The next step in our process is to select a method in order to convert the DBMS to the correct scenario. Recall that some layers need to be converted to either a clean or found state in order to retrieve trusted evidence from the DBMS. The DBMS needs to be converted to the CFC state in this example. Therefore, the data model and application schema need to be cleaned with our methods.

The remaining sections will discuss a variety of combinations of how to reach the CFC state. The DBMS will first be converted to a CFF scenario to clean up the data model. The method selected to achieve the CFF state, or more simply stated - clean the data model layer, is to make a clean installation of the DBMS on a new machine and copy all the other found layers over. This will result in a clean data model installation with a found data dictionary, application schema and application data layer. This method of achieving a clean data model was explained in theory in chapter 5.

Other methods will also be discussed in the below sections. Thereafter the DBMS will be ready to be converted to a CFC scenario by making use of data dumps, as discussed in chapter 7.

8.3.5 Implementing Methods to Convert Environment

After selecting a method to convert the state of the abstract layers to a clean or found environment respectively, we need to implement those methods without altering the evidence of the DBMS.

The approach that will be followed to successfully set up the investigation environment is divided into the following steps:

1. The DBMS stored on the drive of a computer has to be divided into the four abstract layers in order for us to make use of combinations of the DBMS forensic matrix. This can

be done by dividing the folders of the DBMS installation into the appropriate abstract layer or even dividing the contents of the folders into layers, where required.

2. The various folders, files or data have to be extracted from one DBMS installation and combined into another DBMS installation, as required by the scenario from the matrix.

8.3.5.1 Step 1 - Divide Database by Abstract Layer

The definition of each abstract layer needs to be applied to the DBMS that is being used during the forensic investigation which means that the abstract layers must now divide the DBMS installation into four distinctive parts. In chapter 4 the dividing lines between the abstract layers were discussed. Here they need to be determined for the PostgreSQL DBMS.

A forensic investigator cannot allow parts of layers to become mixed up when making use of this method. For example, psql has a data folder that hosts data dictionary, application schema and application data structures. These structures need to be divided according to the abstract layers to avoid a situation where some information of one abstract layer is collected along with information of another abstract layer by accident during the forensic collection process. This will need to be done for each DBMS because the structure of each DBMS is different, but the definitions of the abstract layers remain generic and independent of the DBMS. The psql DBMS has a data subdirectory which the psql documentation refers to as ‘the data dictionary’.

Although this is not inconsistent with the database abstract layers in this document, we still need to separate the application schema and the application data that are located within this data subdirectory. Literature available on the file structures of the psql DBMS as well as some practical testing has revealed that the application data is stored in the data/base/ subdirectory. The application schema is stored in the base/global/ subdirectory. It is also believed at this stage that some of the application schema may reside in the base directory along with the application data. The rest of the data subdirectory contains data dictionary info which includes connectivity details, database storage directories etc. The rest of the

files in the psql installation folder are part of the data model which is the source code that is used to construct the DBMS.

Alternatively, a dump of a database can be used to extract application schema and application data structures. A dump_all can be used to extract data dictionary, application schema and application data structures. The data dictionary, application schema and application data can then be separated further by manipulating the dump script that has been generated. This solves the first step of our approach by dividing the DBMS by abstract layer.

8.3.5.2 Step 2 - Extract Metadata and Data According to Scenario

After the dividing lines between the abstract layers have been defined for the PostgreSQL DBMS, we may proceed to convert the abstract layers to their required state. In this example we shall convert the data model and application layer to a clean state. This will be done by reconstructing the suspected DBMS onto another DBMS. The reason why the DBMS is reconstructed onto another DBMS is to keep the evidence untouched on the suspected DBMS while converting the abstract layers to the correct state.

The two general ways to extract data from one DBMS to a new DBMS is to either make use of data dumps or to copy the folders and files of the DBMS installation. The advantage of making use of data dumps to extract data from one DBMS to another DBMS is that the output of a data dump is generally in a known text format and layers of the DBMS can be divided by simply doing some text editing on the dump file. The disadvantage of making use of data dumps is that the script or function that constructs the data dump may be compromised to deliver manipulated results. Therefore, the two ways in which the data are extracted should be considered and the best choice made to ensure a clean investigation environment.

When a forensic investigator needs to determine what results have been (or will be) returned on the system being investigated, it is necessary to ensure that all four layers from the system are used during the analysis. This may, for example, be necessary when a user claims that his or her actions have been based on data provided by the system - data that,

with hindsight, are clearly wrong. The forensic investigator who has to determine whether this claim is consistent with the operation of the system needs to reconstruct the DBMS. Recall that changes to any layer may influence the results returned and hence it is necessary to replicate all four layers of the system being investigated to the analysis machine. Note that it may also be possible to perform the examination directly on the original system; in this example we assume the originals are copied and archived, and the investigation is carried out using authenticated copies.

The CFF-scenario is the scenario that has been selected in section 8.3.3 to clean the data model due to the welcome message which is displaying wrong results. Scenario CFF can be used in a situation where the tests or analyses reveal that the DBMS has been compromised but it seems that the data model is dictating the data dictionary, application schema or application data not to reveal critical information or evidence. Therefore, the data dictionary, application schema and application data must be copied from the compromised DBMS, and the data model should be retrieved from a clean install.

The simplest way to solve this scenario is to create insert script data dumps from the compromised DBMS and run the script on a clean install of the DBMS but this will not be correct. As with the previous scenario, the pg dump function is part of the psql data model and therefore it cannot be trusted in this scenario, because we want to be sure that the data model is not compromised. The pg dump may have been compromised to deliver data dumps that hide critical information or evidence. Therefore, we need to find another way to ensure that the data dictionary, application schema and application data that are being copied from the compromised DBMS are not manipulated by the data model abstract layer.

The approach used to solve this scenario is to copy the data directory of the compromised psql DBMS to the new DBMS installation. This folder should be copied onto a computer with a clean installation of the DBMS, the relevant files in the data folder should be replaced by the files from the compromised DBMS and the required configurations should be done to enable the server to run normally. Only two virtual machines will be required for this scenario.

After the data model has been cleaned up, we are now prepared to achieve a CFC scenario. The CFC scenario is very similar to the previous scenario where a data dump will be an incorrect approach to collect the evidence. For this scenario three virtual machines will be required to successfully collect the evidence for an analysis. To solve this scenario the data directory of the compromised psql DBMS will once again be copied after removing the application schema structures from the folder. This data directory will replace the data directory in a clean installation of the psql DBMS on the second virtual machine and the required configuration done on the psql installation. Now we have the same situation as the end result of scenario CFF. Now we must only ensure that the compromised application schema is removed from this scenario. Possible methods that can be used to clean the application schema at this stage are listed in chapter 7.

Here we will make use of an old dump of the database that has been taken before the compromise and referencing the correct application structure from the old database dump. A dump_all will also be made from the second machine in the CFF state. This dump will output data dictionary, application schema and application data structures. Note that on the second machine we may now trust the dumps, because the data model has been cleaned up and the dump script is part of the data model.

The third virtual machine should host a clean installation of the DBMS for a clean data model; the data dictionary will be added from the dump script from the second machine to ensure that the data dictionary remains in the found state; the application schema will then be set up according to the information in old dumps from before the DBMS has been compromised to ensure that the application schema is in a clean state, and finally the application data will be copied over from dump from the second machine. This completes the second step of our approach to extract data and metadata from a compromised DBMS to a clean investigation environment.

8.3.6 Results

In this section it will be illustrated that the alterations of the states of the data model and application schema layers to a clean state have actually cleaned up the compromises that have been made to the DBMS earlier in this chapter. The compromised welcome message should be removed by a clean data model and the compromised column swap needs to be removed by a clean application schema.

```
• ./configure
• make
• make install
• adduser postgres
• mkdir /usr/local/pgsql/data
• chown postgres /usr/local/pgsql/data
• su - postgres
• /usr/local/pgsql/bin/initdb -D /usr/local/pgsql/data
```

Figure 8.5. Code used to compile psql

A clean install of PostgreSQL can be executed in Ubuntu by making sure any previous installation of PostgreSQL has been uninstalled and then compiling the psql source code by making use of the commands in Figure 8.5. To set up scenario CFF we are more selective on what we copy from the compromised DBMS. Triggers, procedures and indexes are deemed to be part of the application schema and will be included in the compromised information that needs to be copied from the first virtual machine to the second virtual machine.

After an analysis it was evident that the data folder of the psql installation holds all the data dictionary, application data and application schema structures. The data folder is then compressed into a tarball file and extracted to the second virtual machine. The server on the second virtual machine has to be stopped and the data folder in the clean psql installation is removed. The compromised data file is now copied into the clean installation

of the second virtual machine. The permissions and owner of the data folder must be granted to the postgres user. The server may now be restarted, and the application data and application schema structures are tested.

This time the *normal welcome message* was displayed because the data model was not compromised. The application schema and application data were compromised, and displayed the swapped columns and wrong values respectively. Hence, this scenario was completed successfully, because the data model was cleaned up but the application schema remained in a found state. The commands in Figure 8.6 were used to complete and test this scenario.


- `su – postgres`
- `cp –r data/ /usr/local/pgsql/ - copy compromised data folder to clean installation`
- `chown –R postgres:postgres /usr/local/pgsql/data – set permissions for data folder`
- `/usr/local/pgsql/bin/postgres -D /usr/local/pgsql/data >logfile 2>&1 & - start server`
- `usr/local/pgsql/bin/psql test – log in to database`
- `- normal welcome message`
- `select * from schema; - view swapped columns;`
- `select * from data; - view wrong values in table`

Figure 8.6. Commands used to complete scenario CFF

Scenario CFC follows the same steps as scenario CFF but requires a couple of additional steps as well as the third virtual machine. In scenario CFF we are sure that the pg dump data model function is clean and can be trusted. Therefore, we use this function to create insert scripts for the data dictionary, application data and application schema on the second virtual machine with the scenario of CFF. Now we need to run these insert scripts on the clean installation of the third virtual machine but we also need to insert a trusted application schema on the DBMS of the third virtual machine. In this experiment the

application schema is known and inserted, but for a real forensic investigation only a trusted application schema can be inserted, as discussed in the previous section.

The tests were successful and the application data were in the same state as in the compromised DBMS on the first virtual machine. This time the welcome message was displayed normally and the columns were not swapped anymore. The command in Figure 8.7 below was used to make the pg dump for the insert scripts.



```
pg_dump --inserts test
```

Figure 8.7. Command used to generate insert scripts

Note that the example of compromising the welcome message was useful to practically illustrate how the data model can be compromised but it is important to note that we converted the data model to a clean state and thereby removed the welcome message. What does this mean? We certainly do not want to *remove* evidence from the DBMS by cleaning up the data model.

Here it is important to remember that the cleaning up of the data model serves multiple purposes:

1. Remove the malicious code (or other compromise) to confirm that the data model has been corrupted;
2. remove the malicious code to reveal evidence that is hidden by the malicious code; or
3. understand the extent of the compromise on the data model.

There are many instances where the results of the compromised data model can be used as evidence. It is up to the investigator to decide when to clean up a layer to extract evidence or use the layer as it has been found to extract evidence.

8.4 CONCLUSION

This chapter discussed an approach of how all layers of the DBMS could be collectively viewed in a forensic context when conducting a forensic investigation on a compromised DBMS. Chapter 8 combined the theory discussed in previous chapters to solve the problem of this study which is to determine how one finds the appropriate combination of clean and existing layers for a forensic investigation on a compromised DBMS while ensuring the integrity of the forensic results. This was done by first discussing a process which the forensic investigator could utilise during a forensic investigation on a DBMS. Thereafter the process was practically implemented to illustrate how the process worked and to illustrate that the ideas in this paper were not limited to theory but could be practically applied.

It is no secret that conducting a forensic investigation on a DBMS is complex and chapter 8 deals with this complexity. However, the process and methods presented in chapter 8 provide a formal way to deal with forensic challenges on compromised DBMSs. The authors of this study believe that this process and methods can be applied to a wide range of forensic scenarios that relate to DBMSs.

CHAPTER 9 CONCLUSION

This study makes it possible for a forensic investigator to conduct a forensic investigation on a compromised DBMS. It seems that most investigations are done on DBMSs without considering the notion that the DBMS might be compromised. It is argued that the forensic investigator should contemplate the idea that a DBMS may be compromised and sources of evidence such as log file entries and query results may be incorrect. The DBMS can now be seen in a new light, because the DBMS is just a compilation of code, metadata and data which an attacker can manipulate in numerous ways.

This study assisted the investigator in understanding how to approach an investigation on a compromised DBMS and ultimately identify evidence on a compromised DBMS. An investigation on a DBMS which had not been compromised was covered in this study and required a much simpler approach.

This study also contributed to the database forensics field which is a young field with little information available on how to conduct forensic investigations on a DBMS.

In this study we modified the internal structure of the DBMS to prove that various “hacks” were possible within a DBMS. The code of the DBMS was changed to throw out incorrect welcome messages; it was illustrated how to set up a rootkit on a DBMS; columns were swapped in metadata; operators of the DBMS were altered to deliver incorrect results etc. The clean and found forensic environments were introduced due to the fact that the live and post-mortem forensic environments did not solve the challenge compromised DBMSs had presented. Arguments were provided why the clean and found environments would be useful for the data model layer, data dictionary layer and the application schema layer respectively.

Methods were introduced on how the data model layer, data dictionary layer and application schema layer could be converted to a found or clean state. A forensic study was done on each abstract layer of the DBMS. The DBMS forensic matrix was introduced to

simplify the manner in which an investigator approached a compromised DBMS. Risk profiling was applied to compromised DBMSs in order to determine what the risk of retrieving false evidence from the DBMS entailed. A process was presented which assisted the forensic investigator in identifying trusted evidence from a compromised DBMS. Finally, a practical study was completed on a real DBMS to prove that our approach and methods could be used in practice.

The problem statement of this study is: How does one find the appropriate combination of clean and existing layers for a forensic investigation on a compromised DBMS and ensure the integrity of the forensic results?

This question was answered by dividing the DBMS into four layers according to the ANSI SPARC model, discussing and illustrating the layers in a forensic context, combining the layers to enable a forensic investigation on a full DBMS, and creating a process which an investigator could apply to combine clean and found layers to ultimately conduct an investigation on a compromised DBMS.

Two papers were published with this study. One paper [101] was based on conducting a forensic investigation on the data model of the DBMS and assisted in constructing chapter 5 of this study. The other paper [102] was based on assembling metadata of various layers of the DBMS in order to conduct a forensic investigation on the DBMS and assisted in constructing chapter 8 of this study.

Although this study contributes to the relatively young field of database forensics, it does not solve all the challenges that face database forensics. Our attention has been focused on solving database forensics on a DBMS which has been compromised. However, there are other areas of database forensics that have not been explored and remain undiscovered which are not part of this study.

An explored area of database forensics which is not within the focus of this study includes recovering evidence from a DBMS which has been destroyed, although we have touched

on this subject briefly. This study has not focused on analysing the effects of the changes that occur on a DBMS from the time that the event of forensic interest occurs until the commencement of the forensic investigation.

Being an engineering study in nature, we have not either focused our attention on court procedures and court deliverables, even though we have ensured our arguments are of a nature to hold its own in a court of law.

An undiscovered area of the database forensics field on which we have not focused is in a sense the inverse of this study. Our study has focused on transforming the layers of the database to a state where the layer provides the investigator with a true reflection of what is in the DBMS. A further study may need to be conducted where the data or metadata of a DBMS are changed in order to view how a compromised layer interprets the change in data or metadata. This inverse form has been discussed in this study but has not been our core focus. One may argue that the two forms are similar and both can be solved by making use of the methods within this paper; however, this statement remains to be proven.

Future work in the field of compromised DBMSs that stems from this study includes a comprehensive test study which analyses the methods and processes proposed in this study. This study has tested the methods to a certain extent but further testing is required on DBMSs. The PostgreSQL DBMS has been used to test our methods and processes, and it will be useful to test the same methods and processes on other DBMSs like Oracle, MySQL etc. This study has focused on the evidence identification process of database forensics but there are several stages of the forensic process that may need to be explored in a forensic context in future. The stages of the forensic process that remain to be explored in the context of database forensics are collection of evidence after identification, transportation, reconstruction, analysis and presentation in court.

REFERENCES

- [1] C.J. Date, SQL and Relational Theory. Sebastopol, U.S.A: O'Reilly Media, 2009.
- [2] A. Basta, M. Zgola, D. Bullaboy and T.L. Whitlock, Database Security. Boston, USA: Course Technology, 2012.
- [3] C. Prorise and K. Mandia, Incident Response & Computer Forensics, 2nd ed. Osborne, U.S.A: McGraw-Hill, 2003.
- [4] M. Meyers and M. Rogers, "Digital Forensics: Meeting the Challenges of Scientific Evidence" in *Proceedings of the IFIP International Conference on Digital Forensics*, 13-16 February, 2005, Orlando, USA. Sensors, M. Pollitt and S. Shenoi, vol.1, IFIP 11.9. Springer, 2006, pp. 43-50.
- [5] G. Mohay, Computer and Intrusion Forensics, 1st ed. Norwood, USA: Artech House, 2003.
- [6] A&E Television Networks, "Dennis Rader Biography", 2007, <http://www.biography.com/articles/Dennis-Rader-241487?part=1>. Last accessed on 8 August 2010.
- [7] E. Taub, "Deleting may be easy, but your hard drive still tells all", 2006, <http://www.theglobeandmail.com/news/technology/article819202.ece>. Last accessed on 8 August 2010.
- [8] D. Litchfield, The Database Hacker's Handbook, 1st ed. Indianapolis, USA: Wiley Publishing, 2005.
- [9] E. Bertino and R. Sandhu, "Database Security – Concepts, Approaches, and Challenges", *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 1, pp. 2-19, March 2005.
- [10] D. Gollmann, Computer Security. New York, USA: John Wiley and Sons, 2002.
- [11] S. Sumathi and S. Esakkirajan, Fundamentals of Relational Database Management Systems. Berlin, Germany: Springer, 2007.
- [12] J. Vacca, Computer Forensics: Computer Crime Scene Investigation, 2nd ed. Course Technology PTR, 2005.
- [13] E. Cole, Network Security Bible, 2nd ed. New York, USA: John Wiley & Sons, 2009.

- [14] S. Richmond and C. Williams, “Millions of internet users hit by massive Sony Playstation data theft”, 2011, <http://www.telegraph.co.uk/technology/news>. Last accessed on 17 July 2011.
- [15] R. McKemmish, “When is Digital Evidence Forensically Sound?” in *Proceedings of the IFIP International Conference on Digital Forensics*, 2008, Kyoto, Japan. Sensors, I. Ray and S. Sheno, vol. 1, IFIP 11.9. Springer, 2008, pp. 3-15.
- [16] M. Pollit, Computer Forensics: An Approach to Evidence in Cyberspace, *Proceedings of the Eighteenth National Information Systems Security Conference*, pp. 487-491, 1995.
- [17] C. Altheide and H. Carvey, *Digital Forensics With Open Source Tools*. MA, USA: Syngress, 2011.
- [18] F. Cohen, “Fundamentals of Digital Forensic Evidence” in *Handbook of Information and Communication Security*. P. Stavroulakis and M. Stamp, San Jose, USA: Springer, 2010, pp. 789-808.
- [19] J. Wiles and A. Reyes, *The Best Damn Cybercrime and Digital Forensics Book Period*. MA, USA: Syngress, 2007.
- [20] E. Casey, *Handbook of Computer Crime Investigation: Forensic Tools and Technology*. London, UK: Academic Press, 2002.
- [21] M. Zelkowitz, *Advances in Computers: Information Security*. San Diego, USA: Elsevier Inc., 2004.
- [22] B. Carrier, *File System Forensic Analysis*. NJ, USA: Pearson Education, 2005.
- [23] M. Olivier, On Metadata Context in Database Forensics in *Digital Investigations*, vol. 5, pp. 115-123, 2009.
- [24] E. Casey, *Digital Evidence and Computer Crime – Forensic Science, Computers on the Internet*, 2nd ed. California, USA: Elsevier Academic Press, 2004.
- [25] M.L. Gillenson, *Fundamentals of Database Management Systems*, 2nd ed. New Jersey, USA: John Wiley and Sons, 2005.
- [26] S.K. Singh, *Database Systems: Concepts, Design and Applications*. New Delhi, India: Pearson Education, 2006.
- [27] ITL Education Solution Limited, *Introduction to Database Systems 2008*. New Delhi, India: Pearson Education, 2008.

- [28] A.J. Oppel, Database: A Beginner's Guide. New York, USA: McGraw Hill, 2009.
- [29] J.L. Harrington, Relational Database Design: Clearly Explained. Orlando, USA: Academic Press, 2002.
- [30] M. Norgaard, D. Ensor, T. Gorman, K. Hailey, A. Kolk, J. Lewis, C. McDonald, C. Millsap, J. Morle, D. Ruthven and G. Vaidyanatha, Oracle Insights: Tales of the Oak Table. New York, USA: Apress, 2004.
- [31] The PostgreSQL Global Development Group, PostgreSQL 9.0 Official Documentation: The SQL Language. California, USA: Fultus Corporation, 2011.
- [32] M. Spenik and O. Sledge, Microsoft SQL Server 2000 DBA Survival Guide, 2nd ed. New York, USA: Sams Publishing, 2003.
- [33] R. Flannery, Informix Handbook. New Jersey, USA: Prentice-Hall, 2000.
- [34] C. Esculier and O. Friesen, "Conceptual Modelling: A Critical Survey And A Few Perspectives" in *Proceedings of the 1995 IEEE Fourteenth Annual International Phoenix Conference on Computers and Communications*, 28-31 March, 1995, Phoenix, USA. pp. 319-325.
- [35] D. Hay, UML and Data Modelling: A Reconciliation. New Jersey, USA: Technics Publication, 2011.
- [36] R.T. Watson, Database Management: Databases and Organizations, 5th ed. New York, USA: John Wiley & Sons, 2005.
- [37] G. Keith, Principles of Data Management. Swindon, UK: British Informatics Society Limited, 2007.
- [38] C.T. Yu and W. Meng, Principles of Database Query Processing for Advanced Applications. San Francisco, USA: Morgan Kaufmann, 1998.
- [39] R. Limeback, Simply SQL. Collingwood, USA: Sitepoint, 2008.
- [40] P. O'Neil and E. O'Neil, Database: Principles, Programming and Performance, 2nd ed. San Diego, USA: Academic Press, 2001.
- [41] D.M. Kroenke, Database Processing: Fundamentals, Design, and Implementations. New Jersey, USA: Prentice Hall, 1995.
- [42] J.L. Harrington, SQL Clearly Explained, 3rd ed. Burlington, USA: Elsevier, 2010.
- [43] E.F. Codd, "A Relational Model of Data for Large Shared Data Banks" in *Communications of the ACM*, vol. 13, no. 6, pp. 377-387, 1970.

- [44] M. Levene and G. Loizou, *A Guided Tour of Relational Databases and Beyond*. London, UK: Springer, 1999.
- [45] M. Ferreira, N. Fonseca, R. Rocha and T. Soares, *Efficient and Scalable Induction of Logic Programs Using a Deductive Database System in Inductive Logic Programming*. Berlin, Germany: Springer, 2007.
- [46] C. Churcher, *Beginning Database Design: From Novice to Professional*. Berkely, USA: Apress, 2007.
- [47] T.J. Teorey, S. Buxton, L. Fryman, R. Hartmut, T. Halpin, J.L. Harrington, W.H. Inmon, S.S. Lightstone, J. Melton, T. Morgan, T.P. Nadeau, B. O’Neil, E. O’Neil, P. O’Neil and M. Schneider, *Database Design: Know It All*. Burlington, USA: Elsevier, 2007.
- [48] A. Kriegel, *Discovering SQL*. Indianapolis, USA: Wrox, 2011.
- [49] ITL Education Solutions Limited, *Introduction to Information Technology 2011*. New Delhi, India: Pearson Education, 2011.
- [50] R.F. van der Lans, *Introduction to SQL: Mastering the Relational Database Language*, 4th ed. New York, USA: Addison-Wesley, 2006.
- [51] S.S. Lightstone, T.J. Teorey and T. Nadeau, *Physical Database Design*. San Francisco, USA: Morgan Kaufmann, 2007.
- [52] T. Teorey, S. Lightstone and T. Nadeau, *Database Modeling & Design: Logical Design*, 4th ed. San Francisco, USA: Morgan Kaufmann, 2005.
- [53] A. Ali, *Sphinx Search*. Birmingham, UK: Packt Publishing, 2011.
- [54] B. Nevarez, *Inside the SQL Server Query Optimizer*. Los Angeles, USA: Red Gate Books, 2011.
- [55] J. Lewis, *Oracle Core: Essential Internals for DBAs and Developers*. New York, USA: Apress, 2011.
- [56] C. Ray, *Distributed Database Systems*, New Delhi, India: Pearson Education, 2009.
- [57] J. Hornibrook and N. Kulonovsky, “Best Practices: Writing and Tuning Queries for Optimal Performance”, IBM, Tech. Rep. May 2008.
- [58] C. Ritchie, *Database Principles and Design*, 3rd ed. Bath, UK: Cengage Learning, 2008.

- [59] J. Meloni and M. Telles, PHP 6: Fast and Easy Web Development. Boston, USA: Course Technology, 2008.
- [60] E. Fernandez-Medina and M. Piattini, “Designing Secure Databases,” in *Information and Software Technology*, vol. 47, pp. 463-477, November 2005.
- [61] J. Ashcroft, “Electronic Crime Scene Investigation, A Guide for First Responders”, 2001, <http://www.ncjrs.gov/pdffiles1/nij/187736.pdf>. Last accessed on 7 March 2010.
- [62] S. Sumathi and S. Esakkirajan, Fundamentals of Relational Database Management Systems. Berlin, Germany: Springer, 2007.
- [63] E. Bertino and R. Sandhu, “Database Security – Concepts, Approaches, and Challenges” in *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 1, pp. 2-19, March 2005.
- [64] D. Gollmann, Computer Security. New York, USA: John Wiley and Sons, 2002.
- [65] M. Gertz and S. Jajodia, Handbook of Database Security: Applications and Trends. New York, USA: Springer, 2008.
- [66] E. Bertino, G. Ghinita and A. Kamra, Access Control for Databases: Concepts and Systems. Hanover, USA: Now Publishers, 2011.
- [67] R.J. Anderson, Security Engineering: A Guide to Building Dependable Distributed Systems. Indianapolis, USA: John Wiley and Sons, 2010.
- [68] P. Stavroulakis and M. Stamp, Handbook of Information and Communication Security. Leipzig, Germany: Springer, 2010.
- [69] C.T. Leondes, Database and Data Communication Network Systems: Techniques and Applications. California, USA: Elsevier, 2002.
- [70] D. Cherry, Securing SQL Server. Burlington, USA: Syngress, 2011.
- [71] L. Bouganim and Y. Guo, Encyclopedia of Cryptography and Security, 2nd ed. New York, USA: Springer, 2011.
- [72] B. Bordar, “Model Driven Engineering Languages and Systems,” in *8th International Conference on Models*, October, 2005, Montego Bay, Jamaica. Editors: L.C. Briand and C.E. Williams, Springer, 2005, pp. 382-396.
- [73] S. Tiwari, Professional NoSQL. Indianapolis, USA: Wrox, 2011.

- [74] A. Gates, Programming Pig. Sebastopol, USA: O'Reilly Media, 2011.
- [75] S. Garfinkel, "Digital Forensics Research: The Next 10 Years" in *Digital Investigations*, vol. 7, pp. 64-73, 2010.
- [76] E. Chickowski, "Database forensics still in the dark ages." Internet: <http://www.darkreading.com/database-security/167901020/security/attacks-breaches/231300307/database-forensics-still-in-dark-ages.html>, Aug. 5, 2012. Last accessed May 28, 2012.
- [77] K. Fowler, SQL Server Forensic Analysis. NJ, USA: Addison-Wesley Professional, 2008.
- [78] O.M. Fasan and M.S. Olivier, "On Dimensions of Reconstruction in Database Forensics," in *Proceedings of the Seventh International Workshop on Digital Forensics & Incident Analysis (WDFIA)*, 2012, pp. 97-106.
- [79] D. Litchfield, Oracle Forensics Part 1 Dissecting the Redo Logs, 2007, <http://www.databassecurity.com/oracle-forensics.htm>. Last accessed on 17 March 2010.
- [80] D. Litchfield, Oracle Forensics Part 2 Locating Dropped Objects, 2007, <http://www.databassecurity.com/oracle-forensics.htm>. Last accessed on 17 March 2010.
- [81] D. Litchfield, Oracle Forensics Part 3 Isolating Evidence of Attack Against the Authentication Mechanism, 2007, <http://www.databassecurity.com/oracle-forensics.htm>. Last accessed on 17 March 2010.
- [82] D. Litchfield, Oracle Forensics Part 4 Live Response, 2007, <http://www.databassecurity.com/oracle-forensics.htm>. Last accessed on 17 March 2010.
- [83] D. Litchfield, Oracle Forensics Part 5 Finding Evidence of Data Theft in the Absence of Auditing, 2007, <http://www.databassecurity.com/oracle-forensics.htm>. Last accessed on 17 March 2010.
- [84] D. Litchfield, Oracle Forensics Part 6 Examining Undo Segments, Flashback and the Oracle Recycle Bin, 2007, <http://www.databassecurity.com/oracle-forensics.htm>. Last accessed on 17 March 2010.

- [85] D. Litchfield, Oracle Forensics Part 7 Using the Oracle System Change Number in Forensic Examinations, 2007, <http://www.databassecurity.com/oracle-forensics.htm>. Last accessed on 17 March 2010.
- [86] P. Finnigan. “Oracle Forensics,” http://www.petefinnigan.com/Oracle_Forensics.pdf, Dec. 12, 2007. Last accessed on July 15, 2012.
- [87] P.M. Wright, “Oracle Database Forensics using Logminer,” 2005, <http://www.databassecurity.com/dbsec/OracleForensicsUsingLogminer.pdf>. Last accessed on 3 July 2012.
- [88] K. Fowler, Forensic Analysis of a SQL Server 2005 Database Server, April 2007, http://www.sans.org/reading_room/whitepapers/application/forensic-analysis-sql-server-2005-database-server_1906. Last accessed on 3 July 2012.
- [89] N. Beebe and J. Clark, “Dealing with Terabyte Data Sets in Digital Investigations” in *Proceedings of the IFIP International Conference on Digital Forensics*, 13-16 February, 2005, Orlando, USA.
- [90] P. Fruhwirt, M. Huber, M. Mulazzani and E.R. Weippl, “InnoDB Database Forensics” in *Proceedings of the IEEE International Conference on Advanced Information Networking and Applications*, pp. 1028-1036, April, 2010.
- [91] O.M. Fasan and M.S. Olivier, “Reconstruction in Database Forensics”, presented at the IFIP WG 11.9 International Conference on Digital Forensics, Pretoria, South Africa, January 2012.
- [92] S. Friedberg, “Tool Review – Remote Forensic Preservation and Examination Tools” in *Digital Investigations*, vol. 1, pp. 284-297, 2004.
- [93] G. Grispos, T. Storer and W. Glisson, “A Comparison of Forensic Evidence Recovery Techniques for a Windows Mobile Smartphone” in *Digital Investigations*, pp. 1-14, 2011.
- [94] S. Kedar, Database Management System, Pune, India: Technical Publications, 2009.
- [95] R. Koen and M.S. Olivier, “An Evidence Acquisition Tool for Live Systems” in *Advances in Digital Forensics*, vol. 5, pp. 323-334, 2009.
- [96] P. Rob and C. Corornel, Database Systems: Design, Implementation, and

- Management. USA: Course Technology, 2007.
- [97] PostgreSQL Documentation. “pg_database”
<http://www.postgresql.org/docs/9.1/static/catalog-pg-database.html>. Last accessed on August 22, 2012.
- [98] D. Russel and J. Gennick, Oracle DBA Checklist. Sebastopol, USA: O’Reilly & Associates, 2001.
- [99] J. Dyche, e-Data: Turning Data into Information with Data Warehousing, New Jersey, USA: Addison-Wesley, 2000.
- [100] ANSI/X3/SPARC Study Group, “Database Management Systems: Interim Report” in *ACM SIGMOD Bulletin*, vol. 7, no. 2, 1975.
- [101] H.Q. Beyers, M.S. Olivier and G.P. Hancke, “Arguments and Methods for Database Data Model Forensics” in *Proceedings of the Seventh International Workshop on Digital Forensics & Incident Analysis (WDFIA)*, pp. 139-149, 2012.
- [102] H.Q. Beyers, M.S. Olivier and G.P. Hancke, “Assembling the Metadata for a Database Forensic Investigation” in *Advances in Digital Forensics: Proceedings of the 7th IFIP International Conference on Digital Forensics*, 4 February 2011, Springer, New York.
- [103] B. Panda and J. Giordano, “Reconstructing the Database After Electronic Attacks” in *Database Security: Status and Prospects: Proceedings of the 12th IFIP WG 11.3 Working Conference on Database Security*, 15 July 1998.
- [104] M. Jakobsson and Z. Ramzan, *Crimeware: Understanding New Attacks and Defences*. Boston, USA: Pearson Education, 2008.
- [105] A. Kornbrust, “Database Rootkits: Presented at Black Hat Europe,”
<http://www.red-database-security.com/whitepaper/presentations.html>, March 29, 2005. Last accessed on November 4, 2011.
- [106] E. Casey, *Handbook of Digital Forensics and Investigations*, California, USA: Elsevier, 2010.
- [107] L. Daniel, *Digital Forensics for Legal Professionals: Understanding Digital Evidence from the Warrant to the Courtroom*, Waltham, USA: Elsevier, 2012.
- [108] R. Overill, M. Kwan, L. Chow, P. Lai and F. Law, “A Cost-Effective Model for Digital Forensic Investigations” in *Proceedings of the 5th Annual IFIP WG 11.9*

- International Conference: Advances in Digital Forensics*, 28 January 2009, New York, USA: Springer, 2009.
- [109] W. Curtis Preston, *Backup and Recovery*, Sebastopol, USA: O'Reilly Media, 2007.
- [110] D. Khun, C. Kim, and B. Lopuz, *Linux Recipes for Oracle DBAs*, New York, USA: Apress, 2009.
- [111] R. Greenwald, R. Stackowiak, M. Alam, and M. Bhuller, *Achieving Extreme Performance with Oracle Exadata: Best Practices for Deployments in Enterprise Datacentres*, New York, USA: McGraw-Hill, 2011.
- [112] R.M. Riordan, *Designing Effective Database Systems*. Boston, Massachusetts, USA: Addison-Wesley Professional, 2005.
- [113] C. Coronel, S. Morris and P. Rob, *Database Systems: Design, Implementation, and Management*. Boston, USA: Cengage Learning, 2009.
- [114] E. Oz, *Management Information Systems*, 6th ed. Boston, USA: Course Technology, 2009.
- [115] B. Thomas, *OCA: Oracle Database 11g Administrator Certified Associate Study Guide*. Indianapolis, USA: John Wiley and Sons, 2010.
- [116] P. Ward, G. Dafoulas, *Database Management Systems*. London, UK: Thomson Learning, 2006.
- [117] MySQL AB, *MySQL Administrator's Guide and Language Reference*, 2nd ed. Uppsala, Sweden: MySQL Press, 2006.
- [118] D. Schneller and U. Schwedt, *MySQL Admin Cookbook*. Birmingham, UK: Packt Publishing, 2010.
- [119] MySQL Forge, "MySQL Internals Custom Engine", 2010, http://forge.mysql.com/wiki/MySQL_Internals_Custom_Engine. Last accessed on 2 May 2012.
- [120] Sideris Courseware Corp., *Data Modelling: Logical Database Design*. Newton, USA: Sideris Courseware Corp., 2011.
- [121] J. Sammons, *The Basics of Digital Forensics*. Waltham, USA: Syngress, 2011.
- [122] S. Davidoff and J. Ham, *Network Forensics: Tracking Hackers Through Cyberspace*. New Jersey, USA: Prentice Hall, 2012.

- [123] L. Volonino and R. Anzaldua, *Computer Forensics for Dummies*. Indianapolis, USA: Wiley & Sons Publishing, 2008.
- [124] A. Hoog, *Android Forensics: Investigation, Analysis, and Security for Google Android*. Waltham, USA: Syngress, 2011.
- [125] A. Hoog and K. Strzempka, *iPhone and iOS Forensics*. Waltham, USA: Syngress, 2011.
- [126] R. Jones, *Internet Forensics*. New York, USA: O'Reilly Media, 2005.
- [127] A. Jacot, J. Miller, M. Jacot and J. Stern, *JD Edwards EnterpriseOne: The Complete Reference*. New York, USA: Oracle Press, 2008.
- [128] D. Kreines, *Oracle Data Dictionary Pocket Reference*. Sebastopol, USA: O'Reilly Media, 2003.
- [129] R.J.T. Dyer, *MySQL in a Nutshell: A Desktop Quick Reference*, 2nd ed. Sebastopol, USA: O'Reilly Media, 2008.
- [130] P. Wright, "Using Oracle Forensics to Determine Vulnerability to Zero-day Exploits", February 2007. <http://www.sans.org/> Last accessed on 17 March 2010.
- [131] P. Wright, "Oracle Forensics in a Nutshell", March 2007, <http://www.databasesecurity.com/oracle-forensics.htm>. Last accessed on 17 March 2010.
- [132] D. Litchfield, *The Oracle Hacker's Handbook, Hacking and Defending Oracle*, 1st ed. Chichester, UK: John Wiley & Sons, 2007.
- [133] J. Sammons, *The Basics of Digital Forensics: The Primer for Getting Started in Digital Forensics*. Waltham, USA: Syngress, 2012.
- [134] A. Reyes, R. Britton, K. O'Shea and J. Steel, *Cyber Crime Investigations: Bridging the Gaps Between Security Professionals, Law Enforcement, and Prosecutors*. New York, USA: Syngress, 2011.