An Approach to Examine the Metadata and Data of a Database Management System by making use of a Forensic Comparison Tool

Hector Beyers
Department of Computer Engineering
University of Pretoria
Pretoria, South Africa

Martin Olivier
Department of Computer Science
University of Pretoria
Pretoria, South Africa

Gerhard Hancke
Department of Computer Engineering
University of Pretoria
Pretoria, South Africa

Abstract—**This paper will discuss how a forensic comparison tool can effectively assist in a forensic investigation of the metadata and data of a database installation, and an approach to handle the output of the forensic comparison tool in a forensic investigation. The metadata of a psql DBMS installation was compromised to support this statement. The relational database management system was divided into four abstract layers to separate various types of metadata and separate the metadata from the data. These four abstract layers are the data model, data dictionary, and application schema and application data layers. Code was implemented to construct a forensic tool that compares a suspect DBMS installation with a clean DBMS installation. Any discrepancies between the two DBMS installation are reported. The forensic tool considers two types of comparisons namely a file search and dump search. The file search has a three step procedure of (1) checking if all files in both installations are the same, (2) compare the md5 hashes of files that exist in both DBMS installations, (3) and compare the contents of files which are not the same. The dump search makes a dump of both DBMS installations and compares the output. The dump search is particular useful in managing discrepancies found with the file search. This paper proposes a way in which these discrepancies can be handled by considering various outcomes and scenarios. The four abstract layers make it easier to manage a forensic examination after discrepancies were reported by the forensic comparison tool. An approach is discussed on how to deal with add-ons and different versions of DBMS installations. Although the psql DBMS was used for the forensic tool, the concepts in this paper remain independent of DBMS.**

Keywords-component; database forensics; database metadata;

I. INTRODUCTION

Digital Forensics has grown from a relatively ambiguous tradecraft to an essential part of many investigations. Digital forensic tools are now used by examiners and analysts on a daily basis. Development in digital forensic research, tools and processes over the last decade has been very successful [6]. It is therefore odd that so little literature exist on database forensics. Databases have become an essential part of all industries today [3]. The distribution of information and new technologies results in an increase of complex attacks directed to databases [4]. Although a large amount of study has been done on database security and digital forensics, little work has been done on database forensics [1]. This study will provide a practical framework which a forensic examination can be conducted in with the assistance of a forensic comparison tool.

Several forensic tools do exist to enable digital forensic investigations on various platforms such as EnCase, FTK and ProDiscover IR on file systems [7], Safeback on digital media, PeerLab on fileharing, and forensic tools for smart phones [8]. 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. It is not the intention of this study to recreate one of these digital forensic tools. This study will discuss a process to mirror a DBMS and make a use of a forensic tool to assist our forensic examination process.

A database management system (DBMS) consists of metadata and data. The metadata has the potential to influence the data in a much similar way that a file system of a computer can be manipulated to influence the data the file system points to. This characteristic of a DBMS is a field that has not been

researched yet which may deliver an approach for forensic examinations on DBMSs [1].

The DBMS will be divided into four abstract layers to structure the forensic examination. The procedure of setting up a forensic examination environment is discussed. The forensic tool will then be utilized to make the forensic examination more efficient. The metadata and data of the four abstract layers will then be assembled to deliver results in conjunction with the forensic comparison tool.

Section II will discuss how metadata and data can be tampered with to corrupt a DBMS. The concept of assembling metadata and data of a DBMS is discussed in section III. Section IV will discuss the approach we will take to make use of this forensic tool with a forensic examination. Section V will show the practical results we found with this study. Finally we will draw a conclusion in Section VI.

II. Tampering With the Various Layers of the Database Management System

A. The Four Abstract Layers of the Database Management System

The DBMS consists of metadata and data [5]. We divide the metadata and data of the DBMS into four abstract layers in order for us to conduct a forensic examination on a DBMS in a structured fashion. The first abstract layer of the DBMS is the data model. In short, the data model is the source code of the DBMS. The second layer of the DBMS is the data dictionary. The data dictionary is the metadata that constructs all the databases of the DBMS. The application schema includes user created operations that can manipulate data like database triggers, procedures, functions, and sequences, as well as the logical grouping of database objects such as views, indexes and tables. The fourth abstract layer is the application data. The application data is the rows that are stored within the tables of a DBMS. A full explanation on the four layers of the DBMS can be found in [2].

B. Tampering With the Metadata and Data of a Database Management System

The metadata governs the way in which application data is displayed to the user of a DBMS. To change the metadata to portray an image of the data which is not true can be very beneficial to a criminal. Columns can be swopped around by changing the application schema of the database. The source code (data model) of a DBMS can be corrupted to display modified messages. Both these statements have been proven in [2]. There exist various possible ways in which metadata can be modified to corrupt the data.

III. Assembling Metadata and Data for a Database Forensic Examination

A method was developed to assemble the metadata for a forensic investigation [2]. This section will explain how the assembling of metadata and data holds forensic advantages.

A binary matrix from 0000 to 1111 is used to represent the state of the four abstract layers of the DBMS. The sixteen scenario matrix contains all possible configurations of the four abstract layers. The state of 0000 represents the scenario where all four layers of the DBMS are considered to be clean or trusted. The state of 1111 represents the scenario where all four layers of the DBMS are suspect or cannot be trusted. A DBMS on which a forensic examination is conducted is always in an 1111 state when the examination commences. The metadata and data can then be assembled onto another computer to deliver various scenarios (0000 to 1110) and the examiner can analyze the output. In practice not all 16 scenarios will necessarily be useful. There are scenarios that may never be used, and others that will be frequently required in forensic investigations.

Consider the scenario where we have a DBMS to conduct a forensic examination on. The DBMS is thus in state 1111 because we can trust no part of the DBMS when the forensic examination starts. The examiner does not know this yet, but the data model of the DBMS has been changed to display a fake welcome message, and the application schema has been corrupted in such a way that two columns of a table is swapped around. If the DBMS is mirrored onto another computer, the copy will be in an 1111 state. Now we can start to assemble the metadata and data of the copied DBMS to analyze the output of the various states. If a clean installation of the DBMS is installed on the computer, and the suspect data dictionary, application schema and application data are copied to the installation, then we will have a 0111 scenario. The data model is represented by a zero because it is source code from a clean install and the data model can now be trusted. Therefore the compromised welcome message will no longer exist and the original welcome message will be displayed. If the metadata is assembled to a 0001 scenario, then all the abstract layers can be trusted except for the application data. This means that the swopped columns will be ordered in the original way again.

A copy of the DBMS will be mirrored onto another environment where the analysis will be done. There are several advantages to this approach where the live system is not used during the forensic investigation: the DBMS may be compromised to hide valuable evidence, and evidence may be inadvertently destroyed or changed during the forensic investigation [10]. A mirrored copy will eliminate some of these risks, given that a validated mirrored copy is deployed in an environment outside the live system.

IV. Approach

This study will specifically focus on how a comparison tool can be used to assist a forensic examiner in assembling metadata and data for a forensic examination. The study will also discuss the approach which the forensic examiner should follow in assembling the metadata and data. In order to make the forensic process trustworthy, the validation and verification methodology will be used [9] where the forensic examination is divided between data preservation and data analysis. The assembling of the metadata will be designed to preserve data and the forensic comparison tool will assist in analyzing the data.

A. Forensic Comparison Tool

The first step of the approach is to build a forensic comparison tool which can compare every line of data of two DBMS installations. This will enable the examiner to check the differences between a suspect DBMS (1111) and a clean DBMS (0000) installation. The comparison tool will however not be limited to only test these two scenarios, but should be able to test any two scenarios (0000 to 1111). The comparison tool should also be able to compare data dumps from the two DBMSs. This will enable us to test not only the physical data stored on the computer disk, but also the output from the DBMSs.

The PHP coding language will be used to build the comparison tool. A user interface will be set up by making use of html and the smarty template engine. This will be done to separate the PHP functions from the html display code. The PostgreSQL DBMS will be used in this study to test our approach on a real DBMS.

1) Comparing an installation directory.

The comparison tool will take two installation directories as input to compare the installations. The following four step procedure is used to find the differences in the files of the two DBMS installations. A recursive function will be used to loop through the directories and find all the files in all the directories in the two installation folders. The recursive function then yields an array of all the files in the installation directories. Files that exist in the one installation directory, but not in the other, will be reported. The files that do exist in both installations will be compared by calculating the md5 value of the entire file and comparing only these md5 values. The md5 values are calculated in the following way:

$$x_c = md5(filename_c) \quad (1)$$

$$x_s = md5(filename_s) \quad (2)$$

The x represents the output of the md5 function of the clean and suspect DBMSs respectively where the filenames are the same. Only if the md5 values of the files with the same then a *diff* function will be done on the files to determine precisely which lines of the files do not match. The *diff* function runs through every line of a file to check what discrepancies there might be between the two files. The *diff* function takes up a lot of processing time and therefore we will first test if the files are different by making use of the md5 values.

2) Comparing dump files

Another important part of the forensic comparison tool will be to compare the dump files of the two DBMSs. This functionality will enable the forensic examiner to actually test the output of the database as the metadata gets assembled in various ways. This means that now we will not only be able to see that a file in a DBMS installation was compromised, but we will also be able to see what effect this compromise has on the output of the DBMS. The dump file consists of the application schema and application data. We will test the difference between the two dump files by making use of the *diff* function. Once again the directory of the two dump files are required as input.

B. Utilise the forensic comparison tool by assembling the metadata and data

As discussed in the previous section, we can assemble the metadata and data of a DBMS to deliver various scenarios of clean and suspect abstract layers. The forensic tool will enable us to do some testing after we have assembled the data. The forensic tool will also assist in driving the procedure of how the metadata and data should be assembled, based on results from the forensic comparison tool. Essentially this means that an abstract layer can be eliminated if no discrepancies are found in that layer by the forensic comparison tool. The number of possible scenarios can be calculated using the following equation.

$$z = 2^{x-y} \quad (3)$$

In this equation the z is the number of possible scenarios to test after the comparison tool has eliminated some abstract layers from the DBMS. The x denotes the number of abstract layers, and y denotes the number of layers that were eliminated by the forensic comparison tool.

1) Eliminating abstract layers with the forensic comparison tool.

The procedure of eliminating abstract layers by making use of the forensic comparison tool is explained in this section. The suspect DBMS that needs to be examined is required to be mirrored onto the computer which will host the forensic examination. This will be done by compressing the entire installation directory of the suspect DBMS with tarball and decompressing the tar file on the computer that hosts the forensic examination. The integrity of tarball has been tested by generating md5 hash codes of each file in a pgsql installation, compressing and decompressing the directory, recalculating the md5 hash codes of the decompressed directory, and comparing them to the initial hash codes. The test was done with just over 3000 thousand files and all of the hash codes were similar before and after compression.

A clean installation of the DBMS should also exist on the computer. A clean installation does not only mean to have a clean install of the DBMS, it also means to insert the correct data dictionary, application schema and application data into the clean DBMS. This information has to be found from either database design documentation or previous dumps of the database which can be trusted.

We now have two installations of the DBMS on the computer with a state of 1111 and 0000 respectively. We can start to use the forensic comparison tool. The installation directories of the two installations will be entered as input and the forensic comparison tool will start comparing the complete DBMS installations. The output will report all the files that are present in one DBMS installation, but not in the other. The lines of data which are not the same in files (with the same name) from the two installations will be displayed on the interface. We now have a list of all the discrepancies between the two DBMS installations and we need to determine in which abstract layer these discrepancies fall. As a general rule we can say that if an abstract layer has no discrepancy, that abstract layer can be eliminated.

A study [2] has provided us with rules on how to divide the DBMS into four abstract layers. These rules can be applied to any DBMS like PostgreSQL, MySQL, etc. The PostgreSQL DBMS was divided into four abstract layers. We will use the division lines provided in this study as a guideline to determine to which abstract layer a discrepancy belongs to. In short, within a psql installation the source code belongs to the data model and all files that are not within the data directory of the psql installation can be considered to be part of the data model. The data directory of the psql installation contains the data dictionary, application schema and application data. The *data/base/* directory contains the application data. The *data/global/* directory contains the application schema. The rest of the data directory contains the data dictionary. Using this rule we can determine to which abstract layer a discrepancy belongs to.

A second method of determining which abstract layers could be eliminated is to make dump scripts of the two DBMS installations and comparing them. Some of the files in the installation directory of the DBMS installation may be encrypted. A normal comparison of the files will not deliver sensible results. By making use of dump scripts, we can overcome this problem by comparing the output of the DBMSs. The abstract layers in the dump files are the application schema and application data. The application data are all *INSERT INTO* commands in the dump file. These are the rows of data that are entered into the tables of the DBMS. The application schema makes up the rest of the table and can be identified by commands that create various DBMS structures like *CREATE TABLE, CREATE VIEW* etc. If an abstract layer of the dump files is the same, then the layer can be eliminated.

2) Forensic examination with eliminated layers

When we have mirrored the suspect DBMS onto the computer that will host the forensic examination, and installed a clean DBMS on the same computer, we have two installations with the 0000 and 1111 state respectively. We can now eliminate the abstract layers which were found to be the same to make the forensic examination process more effectively. When no layers have been eliminated we will have the following array of scenarios to test:

TABLE I. NO ABSTRACT LAYERS ELIMINATED

Data Model	Data Dictionary	Application Schema	Application Data
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

If the forensic comparison tool eliminates the data dictionary and application schema, the number of scenarios will decrease significantly according to equation 3:

TABLE II. TWO ABSTRACT LAYERS ELIMINATED

Data Model	Data Dictionary	Application Schema	Application Data
0	*	*	0
0	*	*	1
1	*	*	0
1	*	*	1

Now we can start to test the remaining scenarios by changing the state of the 1111 DBMS installation. For instance, in psql we would make a clean install of the DBMS and copy the whole data directory (which includes the data dictionary, application schema and application data) of the 1111 DBMS. Now we will have the state of 0111 which correlates with the second scenario in table 2.

The approach we suggest for the forensic examination is to start testing the data from the data model down to the application data. The reason for this is that the data model influences the data dictionary, application schema and application data; the data dictionary influences the application schema and application data; the application schema only influences the application data; the application data has no effect on the other abstract layers. Therefore it makes the most sense to start by changing the state of the abstract layer which is highest in the abstract layer hierarchy, because this influences the state of the DBMS the most.

3) Application Data as abstract layer or DBMS output

In practice it will be a difficult task to change the application data abstract layer from a suspect state to a clean state. This will entail to get previous data dumps that we trust or from some logging history. To bring the DBMS into such a state can be a complex and time consuming task. Older data dumps that we can trust have the problem that the data will be outdated. The most sensible way to do this will be to get trusted data dumps and then attempt to bring the data up to date, but this will still be a time consuming process which can possibly be required in some instances.

The application data layer can thus initially be used as output when we assemble the metadata while using the forensic comparison tool. We can change the state of the other layers and check to see what influence this has on the data. For example if the application schema was corrupted to hide a column of data, then this will be revealed when we change the state of the application schema from suspect (1) to clean (0). If the forensic examination did not deliver any evidence before changing the state of the application data, then it will be required to do the time consuming process of changing the state of the application data. We anticipate that the application data layer will be much more useful as output mechanism.

C. Handling of different versions and plug in software

As a general rule this study will initially require the versions of the DBMS installations to be the same release. Various versions of the same DBMS software will show a large amount of discrepancies which will incorrectly be reported. This will make it much more difficult to eliminate abstract layers.

Another general rule will be to bring plug in software to the same version on both DBMS installations. This will just make the forensic examination significantly easier. Alternatively if a plug in is installed in its own folder it might not show in the forensic comparison tool, or if it shows the files can be examined to ensure that they have no influence on the DBMS output. This will however not be an ideal environment to do a forensic examination in.

D. Testing

The testing environment that will be set up to test this will have two computers with Ubuntu as operating installed on both of them. The PostrgeSQL 8.4.5 (psql) source code will be compiled on both of the Ubuntu machines. The first psql installation will be compromised by swapping around two columns within the application schema abstract layer. This will compromise this psql installation to show the wrong results associated with the wrong column name. The data model will also be compromised to display an incorrect welcome message. We will then do a real forensic examination by copying the compromised DBMS to the second Ubuntu machine which will host the forensic examination. The copied DBMS will be in an 1111 state and the clean DBMS will be brought into a 0000 state by creating the same databases, tables and other DBMS structures. The correct data will then be inserted into the clean installation.

We will then run our forensic comparison tool to check if we can eliminate any abstract layers which are identical in both DBMS installation directories. We will also make dumps of both DBMSs and compare the output. The elimination process will then start to determine if we can find this compromise within the DBMS by making use of our forensic examination process.

V. RESULTS

This section will discuss the test proceedings and the results obtained from the test. The testing environment was set up successfully and the suspect DBMS was compromised as done in [2]. The directories of the two DBMS installations were given as input to the html interface. The comparison tool took about 18 seconds to run. We anticipate this to be much more for a DBMS with more tables and data than our test DBMS. The comparison of the two DBMS installation directories yielded the following:

- Bison was installed on the one Ubuntu machine and not on the other Ubuntu machine. This showed up in the *Makefile.global* file. This was rectified by installing bison on the second Ubuntu machine.

  Some binary files like */lib/libecpg.a, /lib/libpgport.a and /lib/libpgtypes.a* had differences within them. The *diff* function did not deliver the difference between the files because the files are of a binary type.

- A difference was detected in /share/man/man1/psql.1 where we changed the welcome message from psql (8.4.5) to psql EVIL (8.4.5).

- There was a difference in the binary files in the */data/global/* and */data/base/* directories. Once again these files are in binary format and not in plain text, thus *diff* just notified us that the files are not the same. This is the directories of the application data and application schema.

Next we compared the dump files of the two DBMSs. The comparison of the dump files yielded the following results:

- Since we only created one test database on both DBMSs, the data dictionaries were the same and showed only the one database.

- The application schema was not the same. The order of the two columns (that were swapped) was not the same.

- The order of the insert script was also not the same and different data would be inserted into the two columns.

Therefore we eliminated the data dictionary abstract layer. We then used our approach to first change the data model to see if we get different results. The data model was brought into a clean state as discussed in the previous sections. We now tested a 0111 DBMS against the 0000 DBMS. The forensic comparison tool was run again, but this time there was no discrepancy for the welcome message, but there were still discrepancies for the application data and application schema in the dump script. We can thus draw the following conclusions because we got a different output:

- The data model on the suspect database was compromised to show a fake welcome message.

- Although it may have been the initial reason, the data model is not the reason at this stage that the application schema and application data are different.

Now we eliminate the data model after we document what compromise it has. The next step is to test a clean application schema. We tested a 0101 DBMS against the 0000 DBMS. We made the application schema clean by recreating the tables of the database and inserting the suspect data into this clean application schema. We ran the forensic comparison tool and this time the dump files were exactly the same. We can draw the following conclusions from this result:

- The application schema layer was causing the discrepancy in the application data.

- The application data layer was influenced to show corrupt results by metadata in other layers.

The results drawn from this forensic examination can now be analyzed to determine how this compromise was done.

VI. CONCLUSION

This study researched how a forensic comparison tool can be used to assist in the process of conducting a forensic examination on a DBMS. An illustration was given on how a forensic examination environment could be set up by making use of the Ubuntu operating system and PostgreSQL DBMS. The study also showed how the assembling of metadata and data has forensic advantages to the forensic examiner.

This study proved that the forensic comparison tool can highlight the areas of interest (or areas that have been compromised) and enable the examiner to do a much more effective examination by eliminating parts of the DBMS that are confirmed to be clean. Metadata that has been tampered with can be exploited by this forensic comparison tool. This study provided a practical way of conducting a forensic examination on a DBMS – a rarely researched field.

Future research will entail the test of more scenarios and more database structures. A single scenario was tested to prove that the forensic comparison tool could make the forensic examination process more effective, but more out of the sixteen scenarios have to be tested.

REFERENCES

[1] M.S. Olivier, On Metadata Context in Database Forensics, in Digital Investigations, vol. 5, pp. 115-123, 2009.

[2] H.Q. Beyers, M.S. Olivier and G.P. Hancke, Asssembling the Metadata for a Database Forensic Examination – A Practical Study, in *Proceedings of the IFIP International Conference on Digital Forensics,* 2-4 February, 2011, Orlando, USA.

[3] S. Sumathi and S. Essakkirajan, Fundamentals of Relational Database Management System – Studies in Computational Intelligence, 1st ed. Springer, 2007.

[4] R. Ben-Natan, Implementing Database Security and Auditing, 1st ed. Elsevier, 2005.

[5] J. Dyche, e-Data: Turning Data into Information with Data Warehousing, 1st ed. Addison-Wesley, 2000.

[6] S. Garfinkel, Digital forensics research: The next 10 years, in Digital Investigations, vol. 7, pp.64-73, 2010.

[7] S. Friedberg, Tool review – remote forensic preservation and examination tools, in Digital Investigation, vol. 1, pp. 284-297, 2004.

[8] G. Grispos, T. Storer and W. Glisson, A comparison of forensic evidence recovery techniques for a windows mobile smart phone, in Digital Investigations, pp. 1-14, 2011.

[9] Y. Guo, J.Slay and J. Beckett, Validation and verification of computer forensic software tools – Searching Function, in Digital Investigations, vol. 6, pp.12-22, 2009.

[10] Association of Chief Police Officers, Good Practice Guide for Computer based Electronic Evidence, 2003, http://cryptome.org/acpo-guide.htm. Last accessed on 4 July 2011.