Assembling Metadata for Database Forensics

Hector Beyers, Martin Olivier, Gerhard Hancke

Chapter 7

ASSEMBLING METADATA FOR DATABASE FORENSICS

Hector Beyers, Martin Olivier and Gerhard Hancke

Abstract Since information is often a primary target in a computer crime, organizations that store their information in database management systems (DBMSs) must develop a capability to perform database forensics. This paper describes a database forensic method that transforms a DBMS into the required state for a database forensic investigation. The method segments a DBMS into four abstract layers that separate the various levels of DBMS metadata and data. A forensic investigator can then analyze each layer for evidence of malicious activity. Tests performed on a compromised PostgreSQL DBMS demonstrate that the segmentation method provides a means for extracting the compromised DBMS components.

Keywords: Database forensics, metadata, data model, application schema

1. Introduction

Computers and other electronic devices are increasingly becoming instruments or victims of crimes [10]. After an unauthorized use of a digital system occurs, a digital forensic investigator performs an analysis to determine what has happened on the system for presentation in court. Although database theory and digital forensics are popular research topics, little published work exists on the combination of the two fields, database forensics [7].

The output from a database is a function of the data it contains and the metadata that describes the data in the database. Several levels of metadata manipulate the data, which creates problems for forensic investigations of static data (dead analysis) and live systems (live analysis) [7]. Static data analysis is performed in a clean and reliable environment, but it does not always provide a complete analysis. A live analysis takes

place *in situ*, but with the possibility that the environment (e.g., operating system) can manipulate the interpretation of the data. In a database, the levels of metadata and data need to be trusted to ensure an accurate forensic investigation. This paper describes an experimental method for creating a clean investigation environment by using a combination of the various levels of metadata and data within a database.

The database forensic experiments employ virtual machines running the Ubuntu 10.4 operating system with a PostgreSQL 9.0 installation. Nevertheless, this study has attempted to be as DBMS independent as possible while not compromising on the details of performing database forensics. The results demonstrate the efficacy of the database forensic method and provides a theoretical basis for future research in database forensics.

2. Database Management System Layers

In general, a DBMS consists of four abstract layers: a data model layer, a data dictionary layer, an application schema layer and an application data layer [7].

The data model layer is a simplified representation of complex, real-world data structures [9]. The basic building blocks for all data models are entities, attributes, relationships and constraints. These are constructed and connected according to the design of the type of data model. In practical terms, the data model layer is the source code that assembles the DBMS.

The data dictionary layer is the code that executes database-specific tasks such creating tables, dumping application data and removing users. The data dictionary is usually independent of a query language such as SQL and is specific to a DBMS.

The application schema records the design decisions about tables and their structures. For example, the application schema contains metadata about the tables created by database users [9]. It includes information that identifies the data that users can access, user-created operations that manipulate data such as triggers, procedures, functions and sequences [8], and the logical grouping of database objects into views, indexes and tables [9].

The application data layer refers to the actual data stored within database tables and physically within data files on the database server.

Separating a DBMS into the four abstract layers helps simplify the database forensic process. It enables an investigator to focus a search for evidence in a case and easily harvest evidence from the database.

3. Database Forensics

Extensive research has been conducted in the areas of database theory, database security and digital forensics. Database forensic investigations are often specific to the installed DBMSs [2, 3, 5, 11]. Considerable information is available on DBMS security flaws [6]; these help reveal the types of attacks that are possible and the artifacts that may exist in a DBMS. Despite the prevalence of databases and the fact that database forensics is an important area of digital forensics [1], little research has focused on database forensics.

One method for performing database forensics builds on the similarities between file system forensics and database forensics [7]. File systems and databases both focus on the retrieval of stored data. A file system describes the information stored on a computer; metadata describes the information stored in a database. The output from a database is a function of the data it contains, as well as the metadata that describes the data in the database. This property of databases has significant forensic advantages and is an unexplored area of research [7].

4. Database Forensic Method

This study focuses on the collection phase of the database forensic process. The collection process involves locating the key evidence and maintaining the integrity and reliability of the evidence [4].

The study builds a structured investigation environment using the various layers of DBMS metadata and data in the collection process. The proposed structure is a 4-bit binary string that ranges from 0000 to 1111. The state of each of the four abstract layers (data model, data dictionary, application schema and application data) of the database is represented using a zero or one. A value of zero in a position of the binary string indicates that the corresponding abstract layer of the DBMS is clean; this means that the investigator can trust the layer of the DBMS and that it is uncompromised. A value of one denotes a potentially compromised abstract layer.

A database under investigation corresponds to Scenario 1111 because all four layers are potentially compromised (i.e., the corresponding binary string has ones in all four positions). For example, in a situation where the application data layer of the DBMS might deliver proof of an illegal compromise, the data dictionary could hide the compromise in the application data. In this situation, an investigator should test Scenario 0001 to view the compromised application data with a trusted data dictionary layer. We discuss several collection scenarios and demonstrate

how they can contribute to a structured method for performing database forensic investigations.

Three virtual Ubuntu machines, each with a PostgreSQL installation, were set up to investigate the collection scenarios. One Ubuntu machine contained the compromised DBMS installation that included a database, tables and records, an application schema and application data. The second machine served as the primary analysis machine. The third machine was an additional (optional) platform for use in complex scenarios.

Setting up the investigation environment involved three steps. The first step divided the DBMS into the four abstract layers. This involved dividing the folders of the DBMS installation into the appropriate abstract layers and dividing the contents of the folders into layers where necessary. Depending on the test scenario, the second step copied the potentially compromised layers to the primary analysis machine. The final step was to deliver the results for analysis.

4.1 Database Segmentation

The first step applies the definition of each abstract layer to divide the DBMS installation into layers. The PostgreSQL database used in the tests has a data folder that hosts the data dictionary, application schema and application data structures. An investigator must identify the specific files in the folder associated with each abstract layer. By separating the data folder into the abstract layers, the investigator can avoid collecting information from one layer along with information from another layer. This process must be applied to each DBMS being examined since each DBMS stores data differently. However, the abstract layer definitions are generic (i.e., DBMS independent).

The PostgreSQL DBMS has a data subdirectory that the documentation refers to as the data dictionary [6]. Although this is consistent with the abstract layers, it is still necessary to separate the application schema and the application data that are both located in the subdirectory. An analysis of the file structures in the PostgreSQL DBMS revealed that the application data is stored in the `data/base/` subdirectory while the application schema is stored in the `base/global/` subdirectory. Some portions of the application schema may also reside in the `base/` directory along with the application data. The rest of the `data/` subdirectory contains data dictionary information, which includes connectivity details, database storage directories, etc. The data dictionary also resides in the `bin/` subdirectory, which stores the functions `dropuser`, `create` and `pg_dump`. These functions are examples of data dictionary structures because they manipulate the viewing of the application schema

and application data. The remaining files in the PostgreSQL installation folder are part of the data model, which corresponds to the source code used to construct the DBMS. Alternatively, PostgreSQL can dump the database, allowing for the extraction of the application schema and application data. The application schema and application data can then be separated by manipulating the dump script.

4.2 Metadata and Data Extraction

Our discussion of the second step focuses on four of the sixteen possible scenarios. These scenarios cover the two ways of extracting data from a DBMS into a clean DBMS. The data can be copied either by dumping data or by copying the DBMS folders and files from the file system. The advantage of a data dump is that its output is in a known text format, and dividing the extracted data into the application schema layer and the application data layer is accomplished by editing the dump script. The disadvantage of a data dump is that a compromised dump script can deliver incorrect results. Therefore, an investigator should always consider both ways of extracting data to ensure a clean investigation environment.

The first scenario, Scenario 1111, represents the case where all four layers of the original system are replicated on the test machine. This scenario mirrors the compromised DBMS directly to the second virtual machine on which forensic analysis is performed. The process of mirroring the DBMS should ensure that nothing in the DBMS has changed. One replication approach is to use data dumps to extract the application schema and application data layers, copy the folders for the data model and data dictionary layers, and combine the layers on the second virtual machine. However, this is not effective because, in the case of a compromised data dictionary, a data dump may return compromised results. Therefore, in Scenario 1111, the best replication approach is to copy the complete folder of the compromised DBMS installation to the second virtual machine.

In the second scenario, Scenario 0000, no abstract layers are compromised; all four abstract layers of the database must be available and trusted, which is seldom the case. It is difficult to obtain a copy of the uncompromised application data. A clean DBMS requires a clean install, and the investigator must then create the data model and data dictionary layers. Based on the design documents, it is possible to build a clean application schema layer. However, the most difficult task is inserting clean application data in the clean DBMS. To insert a clean application data layer, the data must come from a known uncompromised source.

For example, data dumps and exports of tables that were saved before the DBMS was compromised can be considered to be clean application data. Using a data dump of the compromised DBMS requires that the investigator confirm that the data in each record is correct. Because of this complication, a forensic investigation of Scenario 0000 will be rare. However, the process could still be used to recover a complete DBMS for a forensic investigation after a compromise.

The third scenario, Scenario 0011, comes into play when the data model and data dictionary do not reveal critical information or evidence, and the forensic investigation should, therefore, focus on the application schema and application data. Investigating Scenario 0011 requires the application schema and application data to be copied to a cleanly installed DBMS. The simplest way to do this is to create an insert script that dumps data from the compromised DBMS and run the script on a clean install of the DBMS. However, as with Scenario 1111, the `pg_dump` function in a potentially compromised PostgreSQL data dictionary could deliver a data dump that hides critical information or evidence. Therefore, a better process is to copy the data directory of the compromised PostgreSQL DBMS – after excluding all data dictionary structures from the folder – to a computer with a clean installation of the DBMS. This replaces the relevant files in the data folder with the files from the compromised DBMS. The final step is to update the DBMS configuration files to enable the server to run normally.

The fourth scenario, Scenario 0001, is similar to Scenario 0011, where a data dump should not be used to collect evidence. This scenario requires three virtual machines to collect the evidence for analysis. The scenario comes into play when analysis reveals that other abstract layers of the DBMS are manipulating the application data. For example, an application schema trigger could corrupt the application data briefly and the data dictionary or data model could be compromised to hide the evidence. In Scenario 0001, the data directory of the compromised PostgreSQL DBMS should be copied after removing the application schema structures from the folder. This data directory replaces the data directory in a clean installation of the PostgreSQL DBMS on the second virtual machine, and the required configuration is performed on the PostgreSQL installation. This places the second machine in the same analysis situation as in Scenario 0011. At this stage, the data dictionary can be trusted because it is part of the clean install on the second machine. Therefore, the data dictionary function `pg_dump` can be used to create insert scripts for the application data. All application schema information should be removed from these insert scripts before the scripts are executed on the third virtual machine. The third machine should host

```
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
        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 1. Configuration of the compromised DBMS.

a clean installation of the DBMS and the application schema should be set up in advance. This means that the databases, tables, indexes, triggers, etc. come from a trusted source. This trusted source could correspond to the database design documentation for the application schema, scripts that build the application schema from a previous trusted dump, or a confirmed application schema from the investigated insert scripts. Finally, the insert scripts for the application data may be executed, enabling the compromised application data to be inserted into the DBMS with a clean data model, data dictionary and application schema.

5. DBMS Tests

In order to test the four scenarios, a small database was created and populated. Changes were made to each of the four database levels to represent compromises. A forensic copy was created for Scenario 0011, which contained the changes made to the application data and application schema layers, but not the changes made to the data model and data dictionary layers. This enabled us to confirm that the forensic copy operated as expected.

Figure 1 displays the commands used to configure the compromised DBMS. Compromising the data model layer involved changing the wel-

```
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 2. Commands used to compromise the application schema.

come message in the PostgreSQL source code, recompiling and then reinstalling the DBMS. Thus, upon logging in, a user would see the compromised welcome message.

The application schema compromise involved swapping two column names in the `pg_attribute` table; this causes a select query on the named first column to return values from the second column. Figure 2 shows the code used to compromise the application schema.

Compromising the application data involved inserting incorrect values into a table. The compromised data model, application schema and application data helped identify whether or not a compromised layer was present.

6. Test Results

The tests used a clean install of PostgreSQL 9.0 running under Ubuntu 10.4. Scenarios 1111, 0001 and 0011 were tested. Scenario 0000, which involves setting up a DBMS without the use of a compromised database, is not discussed in this paper.

```
su - postgres
cp -r pgsql/ /usr/local/   /* copy compromised psql 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 */
    /* view compromised welcome message */
select * from schema;   /* view swapped columns */
select * from data;     /* view wrong values in table */
```

Figure 3. Commands used in Scenario 1111.

Figure 3 shows the script used in Scenario 1111. A copy is made of the entire PostgreSQL installation folder from the compromised first virtual machine. After stopping the server on the second virtual machine, the copied install folder replaces the clean PostgreSQL installation. Before

```
\d salaries                /* check indexes */
select * from pg_triggers; /* check triggers */
select proname, prosrc from pg_catalog.pg_namespace n
        join pg_catalog.pg_proc p on namepace = n.iod
where nspname = 'public';   /* check functions */
```

Figure 4. Commands used to test general DBMS structures.

restarting the server, the script updates the user rights and ownership of the new PostgreSQL data folder.

Tests of the second virtual machine indicated that the compromised welcome message, the compromised application schema and the compromised application data still exist in the copied PostgreSQL data folder. Figure 4 shows the commands used to extract the triggers, functions and index structures from the compromised database.

```
su - postgres
cp -r pgsql/ /usr/local/   /* copy compromised psql 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 */
         /* view normal welcome message */
select * from schema;   /* view swapped columns */
select * from data;     /* view wrong values in table */
```

Figure 5. Commands used in Scenario 0011.

Scenario 0011 is more selective with regard to the data copied from the compromised DBMS. Triggers, procedures and indexes are part of the application schema and are included in the compromised information copied from the first to second virtual machine. Upon analysis, it was evident that the data folder of the PostgreSQL installation holds all the application data and application schema structures. The process shown in Figure 5 is similar to that used in Scenario 0000, except that it focuses on copying the data folder to the second virtual machine. As before, the script stops the second virtual machine server, copies the data folder and sets the rights and ownership before restarting the server. Testing revealed that the application data and application schema structures were corrupted. Upon logging in, the user sees the normal welcome message because the data model comes from a clean install. As expected, the application data and schema displayed the corrupted swapped columns and falsified values.

Scenario 0001 is similar to Scenario 0011, but it requires additional steps as well as the third virtual machine. In Scenario 0011, it is certain that the `pg_dump` data dictionary function is clean and trustworthy. Therefore, this function can be used to create insert scripts for the application data and application schema on the second virtual machine according to Scenario 0011. Note, however, that the application schema delivered by the `pg_dump` function may not be trusted, so only the application data information from the insert script is usable. Therefore, it is important to first to insert a trusted application schema in the DBMS on the third virtual machine.

The test involving Scenario 0001 was successful. The application data was in the same state as in the compromised DBMS on the first virtual machine. Also, the welcome message displayed normally and the application schema was correct.

7. Conclusions

DBMS metadata and data are vulnerable to compromise. A compromise of the metadata can deceive DBMS users into performing incorrect actions. Likewise, a malicious user who stores incorrect data can affect user query results and actions. Dividing a DBMS into four abstract layers of metadata and data enables a forensic investigator to focus on the DBMS components that are the most likely to have been compromised. Tests of three of the sixteen possible compromise scenarios yielded good results, demonstrating the utility of the database forensic method.

While the four abstract layers divide a DBMS into smaller and more manageable components for a database forensic investigation, the boundaries between the data model and data dictionary, and the data dictionary and application schema can be vague for some DBMS structures. Future research will focus on methods for dividing common DBMS structures into the correct abstract layer categories. Also, it will investigate how metadata and data should be assembled in all sixteen scenarios, and identify compromises of DBMS metadata and data.

References

[1] E. Casey and S. Friedberg, Moving forward in a changing landscape, *Digital Investigation*, vol. 3(1), pp. 1–2, 2006.

[2] Databasesecurity.com, Oracle forensics (www.databasesecurity.com /oracle-forensics.htm), 2007.

[3] K. Fowler, Forensic analysis of a SQL Server 2005 Database Server, InfoSec Reading Room, SANS Institute, Bethesda, Maryland, 2007.

[4] R. Koen and M. Olivier, An evidence acquisition tool for live systems, in *Advances in Digital Forensics IV*, I. Ray and S. Shenoi (Eds.), Springer, Boston, Massachusetts, pp. 325–334, 2008.

[5] D. Litchfield, *The Oracle Hacker's Handbook: Hacking and Defending Oracle*, Wiley, Indianapolis, Indiana, 2007.

[6] D. Litchfield, C. Anley, J. Heasman and B. Grindlay, *The Database Hacker's Handbook: Defending Database Servers*, Wiley, Indianapolis, Indiana, 2005.

[7] M. Olivier, On metadata context in database forensics, *Digital Investigation*, vol. 5(3-4), pp. 115–123, 2009.

[8] Quest Software, *Oracle DBA Checklists: Pocket Reference*, O'Reilly, Sebastopol, California, 2001.

[9] P. Rob and C. Coronel, *Database Systems: Design, Implementation and Management*, Thomson Course Technology, Boston, Massachusetts, 2009.

[10] U.S. Department of Justice, *Electronic Crime Scene Investigation: A Guide for First Responders*, Washington, DC (www.ncjrs.gov/pdffiles1/nij/187736.pdf), 2001.

[11] P. Wright, Using Oracle forensics to determine vulnerability to zero-day exploits, InfoSec Reading Room, SANS Institute, Bethesda, Maryland, 2007.