# Data Hiding Techniques for Database Environments

Heloise Pieterse, Martin Olivier

Chapter 20

# DATA HIDING TECHNIQUES FOR DATABASE ENVIRONMENTS

Heloise Pieterse and Martin Olivier

**Abstract** Databases are widely used today, but the study of forensic analysis in database environments has not been extensive. The lack of research is due mostly to the complex structure of databases and the scarcity of database forensic tools. This has created a wide range of possibilities for data hiding as a means to hinder data discovery during forensic investigations. This paper describes several techniques that can be used to hide data in a database. Although the techniques are evaluated with respect to object-relational databases, they can be applied to any type of database.

**Keywords:** Database forensics, data hiding

## 1. Introduction

Databases provide a means for storing large quantities of data, which can be of interest in forensic investigations [7]. This paper considers several techniques for hiding data in a database. The individual who hides the sensitive data is referred to as the "hider" while the individual who wants to discover the hidden data is referred to as the "examiner." The implementation of data hiding techniques focuses on object-relational databases (ORDs). However, the techniques described in this paper are readily applied to other types of databases.

In general, there are two types of data hiding techniques: data removal and data disguise. The two types of techniques are explored in terms of their application to stored data, data relationships and the database itself.

## 2. Object Relational Databases

Database systems have evolved from hierarchical and network systems [12] to modern object-oriented database management systems and object-relational database management systems (ORDBMSs). At the core of this evolution is the relational database, which has become the most popular database system since it was first developed in the early 1970s [9]. The popularity of the relational database stems from its simplicity and flexibility. Unlike other database systems, it is easy to understand, learn and use. However, a primary limitation is that it does not provide for the use of complex data types and advanced queries. The object-relational database (ORD), also called the object-relational database management system (ORDBMS), is an extension of the relational model [9]. Postgres is one of the earliest ORDBMSs. It extends Postgres by incorporating abstract data types, data of the type "procedure," and rules [8].

### 2.1 Database Forensics

Data is an important commodity [5] that can sometimes be critical [4] (e.g., medical data) or sensitive (e.g., banking system data). Databases are designed to store data and the history of the data [10]. Databases usually record historical data in log files such as the transaction log [3]. Historical data is stored to allow for the recovery of the database after a system failure, to analyze events or to check for compliance with security policies [10]. Remnants of historical data and activities are useful in forensic investigations. Despite the importance of data in investigations, little research has been conducted in database forensics. Much of the research in this discipline focuses on specific database systems such as MS-SQL Server and Oracle [6].

### 2.2 Data Hiding

The concept of hiding data is as old as digital computers and networks [2]. Data hiding embeds data in digital media for various purposes, including identification, annotation and copyright protection [1]. It can be viewed as a secure communication method that enables secret messages to be inserted in plaintext so that they will not draw attention [11].

The most important goal of a data hiding technique is to not restrict or regulate access, but to ensure that the hidden data remains inviolate and recoverable [1]. This implies that any data hiding technique must comply with certain requirements: it must regulate access to the embedded or hidden data, the hidden data must be recoverable and the integrity of the data must be maintained.

In file systems, data hiding techniques typically hide data in the slack space of a hard disk (area between the logical end-of-file and the end of the cluster in which the file is placed) and in digital warrens (created by the absence of a perfect mapping between logical and physical file structures) [2].

These data hiding techniques can be applied to databases, but they have the potential to inconvenience database users. For example, data hidden in the slack space of a hard disk or in a digital warren may be protected from possible detection, but the availability of the data is reduced.

The next section describes some data hiding techniques that allow for the hidden data to be easily restored in the database. This is despite the fact that the techniques are primarily used to prevent the discovery of data that has been hidden in a database.

## 3. ORD Data Hiding

This section focuses on techniques for hiding data in ORDs. In addition to describing the data hiding techniques, it lists their areas of application and explains how they are used in ORDs.

### 3.1 Data Hiding Techniques

Two options are available for hiding data in an ORD. The first option, removal, is to move the data from its original place to a more secure location. The second option, disguise, is to change the appearance of the data, thus decreasing the chances of discovering it.

Data removal can be achieved by transition or by deletion. Data removal by transition moves the data from its current position to a new location. The new location provides the necessary protection – discovering the hidden data is not simple for anyone but the hider.

Data removal by deletion removes the data from a row, column or table. This technique requires that the deleted data be remembered, which may be difficult if a large amount of data is removed.

Data disguise changes the appearance of the data so that it cannot easily be detected. The change of appearance alters the elements of a database that describe the sensitive data that is disguised. The changes should not be obvious to an examiner who views the database.

The subcategories of data disguise include replacement, splitting and encryption. These techniques can be used separately or in combination. The first subcategory, replacement, involves providing a substitute for the sensitive data. The second subcategory, splitting, divides the sensitive data into parts or different groups. The third subcategory, encryption, uses cryptographic techniques to disguise the sensitive data.

In some cases, in order to adequately hide sensitive data, it may be necessary to combine data removal and data disguise.

## 3.2 Database Levels and Data Hiding

Data removal and data disguise can be applied at multiple levels in a database, which includes the stored data, the relationships and the database itself.

Data hiding at the stored data level can involve a single entry in a table, an entire row or column in a table, or a single table. Data hiding at this level differs based on the specific data removal or data disguise technique that is used.

The relationships level is the second level where data hiding techniques can be applied. A relationship "describes an association among entities" [9]. Relationships are an important part of ORDs; they provide valuable information, such as multiplicity that describes the minimum and maximum number of occurrences [3]. Valuable or sensitive information is often provided by or derived from relationships; this makes it necessary to hide the relationships.

The database level is the final level where data hiding techniques can be applied. A database includes the data, tables and system catalog. The system catalog provides information about database data, applications, users and schemas [3]. The description of database data is also referred to as the data dictionary or metadata [3]. To provide adequate protection at the database level, it may be necessary to access and manipulate the system catalog. The system catalog provides direct access to the database and all that it holds; therefore, a data hiding technique should be applied directly to the system catalog.

An essential feature of PostgreSQL databases is the use of schemas [8]. Every database that is created contains one or more schemas, each of which contain tables. Schemas also contain information about other named objects such as data types, functions and operators. Schemas organize objects into logical groups, allowing for easier management. Each new database contains a public schema, which is the default schema. The public schema contains all the tables that are created within the database, unless otherwise specified.

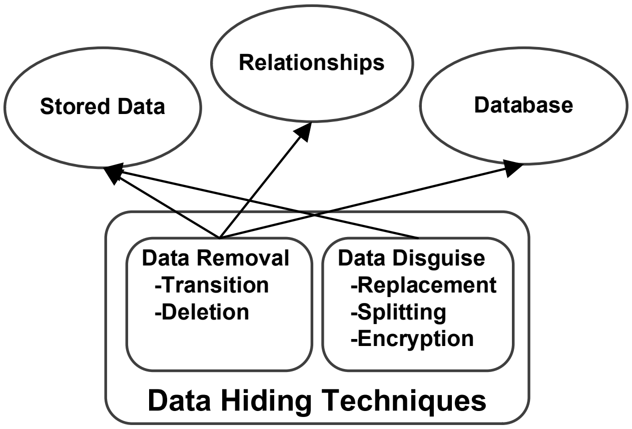

*Figure 1.* Application of data hiding techniques at different levels.

### 3.3 Application of Data Hiding Techniques

As shown in Figure 1, a data removal technique can be applied to the stored data, relationships and the database itself. A data disguise technique can be applied to the stored data.

**Stored Data.** When single rows, columns or even attributes are to be hidden, it is possible to use both categories of data hiding techniques. The first technique, removal, can range from removing a single entry in a table to removing an entire row or column, or even the entire table. The purpose of removing this type of data (e.g., single entry, row or column) is to temporarily hide the sensitive data from possible detection. The removed data should be stored in another location in order to ensure that it can be returned to its original position.

Data that can easily be remembered may be hidden using data deletion. Data deletion can be used when only a single entry needs to be removed from a table.

Protecting sensitive data using a data disguise technique involves making changes to the data stored in a database. This can be accomplished by replacing, splitting or encrypting the database data.

An example of data replacement is to change the name of a column that reads "Telephone Number" to "Fax Number" and so disguise the nature of the data in the column. Another example is to replace a single entry in a table, such as a first name, with a nickname.

An example of splitting is to separate the first names from the last names and not include them in the same column. Another example is to

*Table 1.* PostgreSQL system catalog entries and descriptions.

| System Catalog | Attribute Name | Type | Description |
|---|---|---|---|
| pg_attrdef | adrelid | oid | Table to which column belongs |
| pg_attribute | attrelid | oid | Table to which column belongs |
| pg_attribute | attname | name | Name of column |
| pg_attribute | attisdroppped | bool | Column is dropped |
| pg_attribute | attnum | int | Number of column |
| pg_class | relname | name | Name of table, index or view |
| pg_class | relnamespace | oid | Object ID of namespace |
| pg_class | relfilenode | oid | Name of on-disk file of relation |
| pg_class | relkind | char | Type of table |
| pg_class | relnatts | int | Number of user columns in relation |
| pg_class | relfrozenxid | xid | Permanent transaction ID in relation |
| pg_database | datname | name | Name of database |
| pg_namespace | nspname | name | Name of namespace |
| pg_namespace | nspowner | oid | Owner of namespace |

split specific numbers, such as a telephone number, into its component parts.

Encryption, of course, transforms plaintext data into ciphertext.

**Relationships.** In order to hide data that can be derived from relationships, it may be necessary to remove the relationships. Removing the relationships involves the removal of foreign keys from the tables. The foreign keys can be stored temporarily in unrelated tables. When the relationships are required, the tables can be recombined using a database view.

**Database.** In order to hide a table, schema or database, it is necessary to access and manipulate the system catalog of the ORDBMS. Since the focus of this paper is on the PostgreSQL ORDBMS, the system catalog schema is the system catalog that is considered in detail. In addition to the public and user-defined schemas provided by PostgreSQL, each database contains a pg_catalog table [8]. The pg_catalog table is the system catalog; it contains all the system tables and the built-in data types, functions and operators. The system tables contain all the metadata associated with a particular database. Table 1 shows the attributes of the catalogs that are of particular interest.

When manipulating the PostgreSQL system catalog, the historical data must also be modified. Historical data provides for the recovery of a database should a failure occur. This data is collected in log files and it is the transaction log file that stores all the interactions that occur in a database. Thus, any changes made to the system catalog are recorded in PosgreSQL log files. In the case of the Windows Vista operating system, the files are located in the directory: C:\Program Files\PostgreSQL\8.4\data. The `pg_log` folder contains the transaction log files (an example filename is `postgresql-2011-07-20_213918.log`).

Removing the log files is a solution, but the absence of the files is an indicator that data has been modified. Instead of removing the log files, a better solution is to change the PostgreSQL configuration file (filename: `postgresql.conf`) `log_destination` field from "stderr" to nothing. By removing this value, no data is recorded in the log files and they remain empty.

A sample PostgreSQL database is used to illustrate the the methods for hiding data using the system catalog tables. The sample database contains a public schema and a schema named `test_schema` that is used when manipulating catalog tables. Two tables, `table_one` and `table_two`, are defined in the sample database.

**Hiding Columns.** The first technique focuses on hiding a column of a table. This technique is referred to as Data Hiding Technique 1 or DH1. The reason for moving a particular column is to hide the data in the column. By placing the column in another table, the data is still available, but it is meaningless in the new table.

To manipulate columns, which contain the attributes of a table, it is necessary to manipulate the `pg_attribute` table. To accomplish this, the object identifier (oid) of the two tables (the table containing the column to be moved and the table to which the column is to be moved) must first be identified. For example, the query:

```
SELECT attrelid FROM pg_attribute
  WHERE attname = ‘surname’;
```

might return an oid of 24600. Then, by performing an UPDATE SQL statement on the `pg_attribute` catalog table, the `attrelid` attribute can be changed to contain the oid of the column that must be moved. For example,

```
UPDATE pg_attribute SET attrelid = 1234
  WHERE attrelid = 24600;
```

moves the column to oid 1234. To return the column to its original table, the same UPDATE SQL statement can be used by swapping the oids.

There are some drawbacks associated with removing a column from one table to another. The column intended to be moved must be specified by the column name via the attname attribute of pg_attribute. Also, it is not possible to perform INSERT SQL statements into a table when a column is not in its original table. Therefore, the column must be returned to its original table before any changes can be made to the table. After the changes have been made to the table, the column containing the sensitive data can once again be removed from the table and placed in another table.

It is also possible to remove all the columns of a single table simultaneously. The only difference is that the column name, specified by the attname attribute of pg_attribute, must not be included in the UPDATE SQL statement. An example is:

```
UPDATE pg_attribute SET attrelid = 1234
  WHERE attrelid = 24600;
```

The drawback associated with removing all the columns of table is that the resultant table becomes empty. This might look suspicious to an examiner viewing the database.

A second column data hiding technique is to move columns into a non-existent table. This technique is referred to as Data Hiding Technique 2 or DH2. It uses the same process as DH1, but instead of a valid destination attrelid attribute, it uses one that is not present in a table in the database.

The reason for moving a single column or multiple columns to a non-existent table is to provide complete protection of the data in the columns. The advantage of this technique over DH1 is that the sensitive data is completely removed from the database. It is, therefore, not possible for an examiner to stumble upon the data in another table by mistake. The drawback of DH2 is that it is inconvenient to move the columns between tables if an update to the table has to be made or if the data contained in the column is required.

Data Hiding Technique 3 or DH3 hides data without removing the column(s) from the table. This is achieved by changing the value of the attisdropped attribute contained in the pg_attribute system catalog table. The attisdropped attribute contains a Boolean value to identify whether or not a column has been dropped and is no longer valid. If the column is present in the table and has not been dropped, then the value of attisdropped is false (f). If the attisdropped value is changed from false (f) to true (t), then the column is no longer present in the table. An example is:

```
UPDATE pg_attribute SET attisdropped = ‘t’
  WHERE attname = ‘number’;
```

where the column cannot be accessed via standard SQL statements such as SELECT. Although the column does appear to be completely dropped from the table, it can be retrieved by simply changing the `attisdropped` value back to false (f). It is also possible to drop multiple columns from a table using this technique.

The advantage of applying this technique is that only a single value needs to be changed to remove a column from the table. However, it has the same inconvenience as DH1 and DH2 in that the value must be reset before the data can be accessed.

To discover data hidden using DH1, it is necessary to check for inconsistencies in the `attnum` column of the `pg_attribute` system catalog table. An example of an inconsistency is when the numbers are not in sequence in the `attnum` column (e.g., one column is numbered 1 and the next column is numbered 3). The following SQL query can be executed to check for such inconsistencies:

```
SELECT attnum FROM pg_attribute
  WHERE attrelid = 24600
  AND attnum > 0;
```

It may be that there are no inconsistencies in the `attnum` column even though a column might still have been hidden in the specific table. The following SQL query determines whether a column has been moved to a table (thus, the particular column is included by means of the ADD COLUMN or CREATE TABLE function):

```
SELECT attrelid, relnatts
  FROM pg_attribute, pg_class
  WHERE attrelid = relfilenode
  AND attnum > 0
  AND attrelid = 24600;
```

The result of this query needs to be interpreted. When the number of rows in the query does not correspond with the value found in the `relnatts` column, then there is a column in this particular table that does not belong to the table. However, it is not possible to determine the exact column using a query. Therefore, it is up to the examiner to find the column.

To discover a column that is hidden in a non-existent table using DH2, the following SQL query can be executed:

```
SELECT * FROM pg_attribute
  WHERE attrelid NOT IN
  (SELECT relfilenode FROM pg_ class);
```

To discover a column that is hidden using DH3, the following SQL query can be executed:

```
SELECT * FROM pg_attribute
  WHERE attisdropped = ‘t’;
```

We are now in a position to discuss the efficiency of the data hiding techniques. Since only one query is required to discover the data hidden by DH2 and DH3, the efficiency of these techniques is low. On the other hand, DH1 requires more interaction to discover a table that possibly contains hidden data. However, since the exact column cannot be determined by the query, the efficiency of DH1 is high.

**Hiding Tables.** Since tables are defined within the schemas of a database, hiding a table requires moving it from one schema to another. Two options are available: move a table from one schema to another in a single database, or move a table from one schema to a non-existent schema.

Moving a table between schemas (i.e., between the public schema and a user-defined schema) in a single database is accomplished by manipulating the `pg_class` system catalog table. The attributes of importance in the `pg_class` catalog are `relname` (name of the table) and `relnamespace` (oid of the schema containing a specific table). This technique is referred to as Data Hiding Technique 4 or DH4.

The first step is to obtain the oid for the `relnamespace` of the schemas. The following query returns the oid of a specific schema:

```
SELECT relnamespace FROM pg_class
  WHERE relname = ‘table_two’;
```

To move the table from one schema to another, it is necessary to change the oid in the `relnamespace` column of the table that needs to be moved. An example is:

```
UPDATE pg_class SET relnamespace = 24602
  WHERE relnamespace = 2200;
```

This query moves all the tables that are currently located in schema with oid 2200 (public schema) to the user-defined schema with oid 24602. To move only a single table, the name of the table must be included in the query definition. An example is:

```
UPDATE pg_class SET relnamespace = 24602
  WHERE relnamespace = 2200
  AND relname = ‘table_two’;
```

The final possibility is to move a table to a non-existent schema. This technique is referred to as Data Hiding Technique 5 or DH5. Moving the table into a non-existent schema is accomplished by manipulating the `relname` and `relnamespace` attributes in the `pg_class` catalog table. The same procedure as in DH4 is followed with the change that the `relnamespace` attribute must contain an oid of a non-existent schema. Moving more than one table requires executing the single table move for each table. It is also possible to move all the tables from a schema to a non-existent schema. This is done by executing the following SQL query:

```
UPDATE pg_class SET relnamespace = 12345
  WHERE relnamespace = 2200;
```

The advantage of moving the tables to a non-existent schema is that sensitive data in the tables is completely hidden. It is unlikely that a forensic examiner will find this data by chance.

DH4 and DH5 are successful in terms of hiding data because the PostgreSQL commands `\d` (that lists tables, views and sequences) and `\dt` (that lists tables), only show the tables that exist in the public schema. Tables that are hidden in a user-defined schema or in a non-existent schema are not shown by these commands. Therefore, both these data hiding techniques can be considered successful.

In order to discover a table that is hidden in a user-defined schema (DH4) or in a non-existent schema (DH5), the following SQL query can be executed:

```
SELECT * FROM pg_class
  WHERE relnamespace != 2200
  AND relnamespace != 11
  AND relnamespace != 99
  AND relnamespace != 11326
```

This query outputs a list of schemas. To discover a hidden table, the following SQL queries must be executed in sequence (assuming that a table is hidden in the `pg_catalog` schema):

```
CREATE VIEW view1 AS SELECT *
  FROM pg_class
  WHERE relkind = ‘r’ AND relnamespace = 11

CREATE VIEW view2 AS SELECT *
  FROM pg_class
  WHERE relfrozenxid = 649
```

```
SELECT * FROM view1
  WHERE relname NOT IN
  (SELECT relname FROM view2)
```

Detecting the use of DH4 only requires an examination of the output of the first query that lists the hidden table. Therefore, DH4 has a low hiding efficiency. To discover data hidden by DH5, several sub-queries are required and, therefore, the efficiency of DH5 is medium.

**Hiding Databases.** There is one remaining technique that can be used to hide an entire database through data removal. This technique uses the pg_dump function to back-up a PostgreSQL database [10]. It allows the database "dump" to be output in a script or archive file format. A script dump is a plaintext file that contains the SQL commands required to reconstruct the database to its state before it was dumped. The advantage of a script dump is that it can be used to reconstruct the database on other machines or even other architectures and, with some modifications, even on other SQL database products.

## 4. Conclusions

This paper has presented several new techniques for hiding data in database environments. In particular, it demonstrates how sensitive data can be hidden within an ORD using PostgreSQL queries. Since only certain data is hidden, the availability of the other stored data is always maintained. This is a key benefit because availability is an important property of databases.

## References

[1] W. Bender, D. Gruhl, N. Morimoto and A. Lu, Techniques for data hiding, *IBM Systems Journal*, vol. 35(3-4), pp. 313–336, 1996.

[2] H. Berghel, Hiding data, forensics and anti-forensics, *Communications of the ACM*, vol. 50(4), pp. 15–20, 2007.

[3] T. Connolly and C. Begg, *Database Systems: A Practical Approach to Design, Implementation and Management*, Addison Wesley Longman, London, United Kingdom, 2009.

[4] K. Fowler, SQL Server database forensics, presented at the *Black Hat USA Conference*, 2007.

[5] P. Fruhwirt, M. Huber, M. Mulazzani and E. Weippl, InnoDb database forensics, *Proceedings of the Twenty-Fourth IEEE International Conference on Advanced Information Networking and Applications*, pp. 1028–1036, 2010.

[6] M. Guimaraes, R. Austin and H. Said, Database forensics, *Proceedings of the Information Security Curriculum Development Conference*, pp. 62-65, 2010.

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

[8] PostgreSQL Global Development Group, PostgreSQL 8.4.12 Documentation, San Francisco, California (`www.postgresql.org/docs/8.4/interactive/index.html`), 2009.

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

[10] P. Stahlberg, G. Miklau and B. Levine, Threats to privacy in the forensic analysis of database systems, *Proceedings of the ACM International Conference on Management of Data*, pp. 91–102, 2007.

[11] N. Wu and M. Hwang, Data hiding: Current status and key issues, *International Journal of Network Security*, vol. 4(1), pp. 1–9, 2007.

[12] A. Yeung and G. Hall, *Spatial Database Systems: Design, Implementation and Project Management*, Springer, Dordrecht, The Netherlands, 2007.