Database Forensics

Robert Horn
Auburn University
COMP5350
Digital Forensics
(334) 750-7693
reh0009@auburn.edu

ABSTRACT
Database forensics is still a relatively new branch of digital forensics. This can be due to the high complexity of relational database management systems or the major differences between these systems. But the number of security breaches of database systems is increasing, giving incentive to study the subject more closely. In this paper, we discuss areas of importance dealing with database forensics as well as techniques to use during a forensic investigation of a relational database management system with specific focus on Oracle database systems.

Categories and Subject Descriptors
H.2.0 [**Database Management**]: General – *Security, integrity, and protection.*

General Terms
Management, Security

Keywords
Database Security, Database Forensics, Data, Oracle Databases

1. INTRODUCTION
With the growth of the Internet comes an increase in cybercrime. The Internet is widely used for commercial transactions, and digital crime cannot always be prevented. Database forensics is a fairly new branch of digital forensics but is becoming more important field as the digital age progresses. Databases hold very critical information, making them critical assets in a forensic investigation. A challenge with database forensics is that there are many different database systems that require a vast knowledge of the system internals to perform a precise forensic analysis. For the sake of this paper, we will focus on Oracle databases, but will also discuss different items of importance in a forensic investigation of a general DBMS as well as different forensic analysis techniques.

2. TRANSACTION LOGS
Transaction logs (redo logs) keep track of actions executed by a database management system. Transaction logs are primarily used to guarantee that database transactions are processed reliably or to bring the database back to a consistent state after an unexpected system failure. These files can also be used in a forensic analysis since they keep a record of all changes made to the database. In this section we discuss data stored in transaction logs of forensic interest, the lifetime of transaction logs, and methods for analyzing these transaction logs.

2.1 Transaction Log Data
Transaction logs contain a great amount of data that can be used for timelining a forensic investigation such as timestamps of transactions, the command executed by the system, and before and after images of the data. With this information at hand, an investigator can timeline a suspect's actions as well as recover sensitive data and restore the database to a prior state [3]. Each record in a transaction log file is made up of change vectors which describe a change that was made to a single block in the database [5]. However, not all transactions are immediately recorded to the transaction logs. When a transaction is made on the database, a new transaction record is placed in a log buffer and is not written to the log file until a) a certain time interval passes, b) a commit is executed c) the buffer fills or d) on a checkpoint [5].

2.2 Transaction Log Lifetime
The lifetime of a transaction log depends on the DBMS and resource constraints of the disk. Once the log file fills up, it will circulate and start writing over old records. This causes the lifespan of transaction logs to vary depending on the size of the file, frequency of updates, and the space required for an individual transaction. A large system with heavy transaction processing could circulate in a matter of days, whereas other systems could have logs with months' worth of data [3]. Some database systems will archive transaction logs. For example, Oracle databases have a configuration that, when activated, will enable you to create archived copies of online transaction logs and store them in an offline location. This setting is not turned on by default. This archiving feature also doesn't allow the database to overwrite previous logs in the log file until the log has been archived. Therefore there is no loss of previous data [5].

2.3 Transaction Log Analysis
Analyzing transaction logs can help pinpoint corruption of a database system occurred, determining specific actions to be taken for a recovery at the transaction level, performance tuning through trend analysis, and discovering possible data tampering. There are open-source and commercial products that can be used for analyzing transaction logs. LogMiner is a part of Oracle Database and as of Oracle 11g is integrated with Oracle Enterprise Manager to provide an easy to use GUI along with the SQL interface with the DBMS_LOGMNR PL/SQL package [6].

2.3.1 *Viewing Transaction Log data with LogMiner*

LogMiner requires a source database, a mining database which can be the same as the source database, the transaction log files to be used, and a dictionary produced from the source database the log files came from. To get started using LogMiner, you specify the dictionary to use by using the DBMS_LOGMNR_D.BUILD procedure and then specify a list of transaction log files to be used for analysis with the DBMS_LOGMNR.ADD_LOGFILE procedure. Next is starting LogMiner. When you start LogMiner, you can specify how to filter data (by timestamps or SCN value) and data formatting options. Start LogMiner with the DBMS_LOGMNR.START_LOGMNR procedure. This will cause the V$LOGMNR_CONTENTS view to be populated. Once V$LOGMNR_CONTENTS is populated, you can query the view for the data of interest [6]. When searching through the transaction logs, an investigator should have a good time frame that they want to search through to avoid searching through large amounts of irrelevant data.

2.3.2 *Transaction Log Format*

The transaction logs are stored on the file system as physical files, each log file made up of the redo entries or redo records. The online transaction log file header is in two parts. The first details the information about the file such as file type, block size, number of blocks, etc. and the second part contains the information about the database instance itself. The first part we will call the file header and the second part the transaction log header. We will be more concerned with the transaction log header.

2.3.2.1 *Transaction Log Header*

The transaction log header holds more information that we are interested in than the file header. The transaction log header contains information about the database such as the database SID, version, and log times. Looking at Figure 1, we see that the transaction log header contains four different System Change Numbers (SCNs) and their associated timestamps. These SCNs are used by the database system to distinguish different versions of the state of the database. When a data manipulation operation is performed, the database uses the SCN to track this change. These SCNs are also used for restoring the database to the specific "version" of the database state [7].

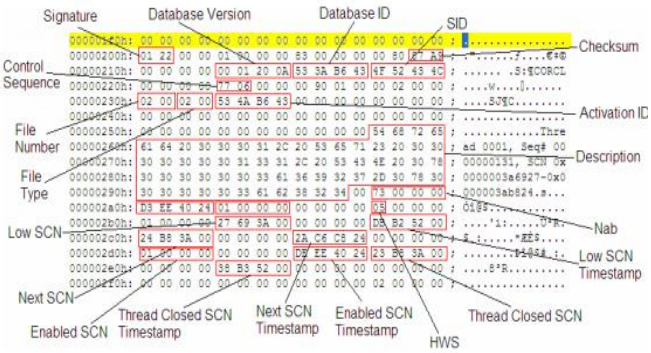

Figure 1. The Transaction Log Binary Format [7]

Each entry has an associated timestamp that is recorded in second. This timestamp is when the entry is written to the log file, not when the event occurred. This will cause there to be a slight difference between when the operation was performed and when the entry was written to the log file. This is an important factor when considering a forensic analysis of the log files. Though DDL operations are automatically committed, therefore the timestamp is accurate. Each timestamp is recorded to the second from midnight on the 1st of January 1988. The timestamp is stored as 32-bit value that is an encoded version of the time since midnight on the 1st of January 1988. The value is encoded as follows: take the year and subtract 1988, multiply this value by 12 and add this to the current month minus 1, multiply the result by 31 and add the current day minus 1, multiply by 24 and add the current hour, multiply by 60 and add the current minutes and then multiply by 60 and add the seconds [7]. For example, if our date and time is 2013-04-3 13:10:45 then the timestamp would be:

2013 – 1988 = 25

25 * 12 + 4 - 1 = 303

303 * 31 + 3 – 1 = 9395

9395 * 24 + 13 = 225493

225493 * 60 + 10 = 13529590

13529590 * 60 + 45 = 811775445

So our date in timestamp form would be 811775445 or 3062B5D5 in hexadecimal. Now that the encoding method is known, the timestamp value can easily be decoded by doing the reverse order of the encoding method and using modulus to grab the individual time unit and dividing to get to the next time unit [7].

Table 1. Operation codes for common change vectors [7]

Operation Code	Change Vector Description
5.1	Undo Record
5.4	Commit
11.2	INSERT on single row
11.3	DELETE
11.5	UPDATE single row
11.11	INSERT multiple rows
11.19	UPDATE multiple rows
10.2	INSERT LEAF ROW
10.4	DELETE LEAF ROW
13.1	Allocate space [e.g after CREATE TABLE]
24.1	DDL

2.3.2.2 *Transaction Log Redo Entries*

A redo entry corresponds to a single SCN and contains all changes for that SCN. The entry has a header and one or more change vectors for the event that occurred in the database. An event could have multiple change vectors, as a given event could affect multiple parts of the database. For example, if a table has an index and an INSERT statement is performed on that table, multiple change vectors will be created. Each change vector has a corresponding operation code for differentiating between change vectors. Table 1 shows the operation code that corresponds to some common change vectors. When an investigator looks through the redo entries they will need to be able to pick out

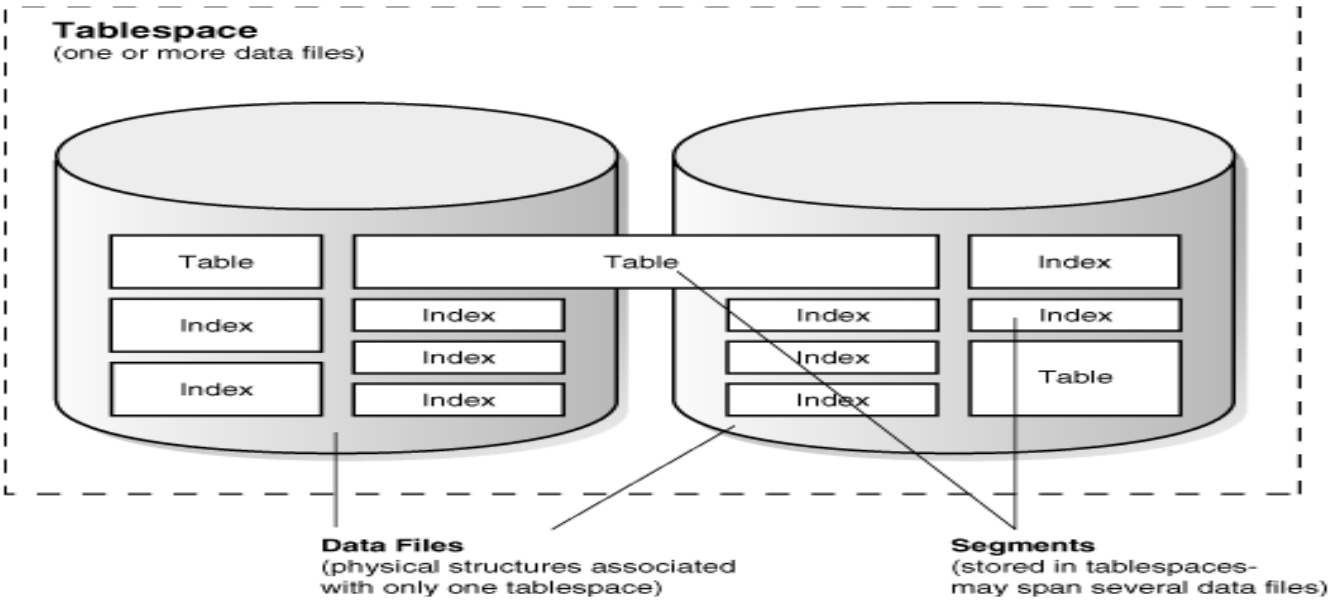

Figure 2. Illustration of the relationship between Tablespaces, Data Files, and segments [8]

entries that are from normal activity and those that are of forensic interest such as from an attack on the system.

3. DATA FILES

Data files are physical paged files on the file system that the database system stores the data from the database tables in. These files have a size limit and it is common for system administrators to add more data files to expand the size of the database when needed. Since these files are stored on the file system, they are subject to the same type of attributes and analysis as a normal file such as encryption, slack space, and operating system requirements and structure.

3.1 The Use of Data Files

The Oracle database system stores individual data and objects in segments, which are stored in space allocated in tablespaces. A tablespace is a logical storage structure that consists of one or more data files. A segment can span across multiple data files, but cannot span across multiple tablespaces [8]. Figure 2 illustrates the relationship between data files and tablespaces. These data files store all the data structures (tables, indexes, etc.) and the data of these structures.

As the amount of data increases on the system, more data files will need to be created to house the new data. An individual data file can either be online or offline (available or unavailable). While data file is offline, the data contained in the data file cannot be accessed until the data file is brought back up online. Data files may be taken offline to perform offline backups, renaming the file, or if there is block corruption. The database automatically takes a data file offline if it is unable to write to the file [8]. It is important for an examiner of a system to remember to check that all of the data files are online to be able to view the data contained within them.

3.2 Data File Structure

When a data file is created for a tablespace, the database allocates the amount of disk space for the data file plus the overhead for the data file header and formatted for use by the tablespace. The space will contain no user data initially, but it is reserved for future data segments of the tablespace. Inside the data file header is the metadata information about the specific data file such as file size, checkpoint SCN, and absolute and relative file number. The absolute file number is in relation to the database while the relative file number is in relation to tablespace. Both uniquely identify the data file.

Within a data file there are multiple extents. An extent is a group of data blocks used for storing specific data which makes up the segment. Within the data file, extents are either used or free. A used extent contains segment data and the free extents are either extents previously used or never used and are available for use. Updates and deletion of objects within the tablespace can cause there to be space that is too small to be reused for new data, referred to as fragmented free space [8].

3.3 Temporary Data Files

Temporary data files are created and used as part of a temporary tablespace which contains objects for the duration of an individual session. These temporary data files are used to store data in hash and sort, and are used to store result set data when sufficient memory is unavailable [8]. Temporary data files do not generate transaction logs and are not recognized by media recovery tools. These data files are typically used to help boost performance and improve efficiency of management operations during sorting procedures such as GROUP BY or ORDER BY clauses used in querying the database.

3.4 Data File Analysis

Analyzing the data files of the Oracle database system can be complicated as the data written to these files are in a format that cannot be read by other programs [8]. But we can analyze other aspects of the data files such as deleted data and how fragmentation is handled by database systems. As mentioned before in section 3.2, over time the space allocated for data files can start to become fragmented from deletion and updates of the data. In most situations, when deletion of data is called, the data is set as free but isn't deleted. This leaves the data recoverable in the

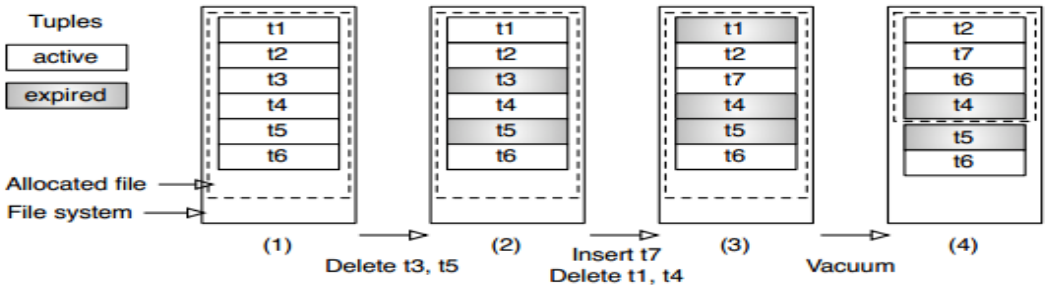

Figure 3. Illustration of table storage for a database undergoing insertions, deletions, and reorganization (vacuum). [3]

space until the space it occupies is reused. The space can only be reused if the new data is able to fit in the free space.

To handle the fragmentation of the data file, the database has a command for reorganizing the space used by the data file. Stahlberg, Miklau, and Levine refer to this command as *vacuum* [3]. When this process is executed, it will reorganize the data file space allowing for more space for new data. But the process does not completely remove all deleted data from the space. Figure 3 illustrates an example of this scenario following a sequence of operations performed on a table stored within the data file [3].

1. The first state (1) shows an initial state of the table containing 6 active records occupying most of the allocated space of the data file [3].
2. The second state (2) shows the state of the table after the deletion of two records, the records remain in the data file space waiting for reuse [3].
3. The third state (3) shows the state of the table after insertion of a new record occupying the space of a previously deleted record, overwriting the previously recoverable data of *t3*. The deletion of two other records also occurs [3].
4. The fourth state (4) shows the state of the table after the reorganization (vacuum) process runs. The active records are reordered, overwriting the previously recoverable record of *t1*. It also reduces the size of the allocated for the file and leaves a copy of the active record *t6* [3].

In the case of updates, the table storage algorithm will leave other traces of recoverable data. If the attribute values of records are to be replaced by smaller values, the table storage algorithm will update the attribute in-place, overwriting the values in the original record. This will leave a partially recoverable piece of data of the previous value. If the attribute is updated with a larger value, then the storage algorithm will mark the original record as deleted and place the new copy of the record with the updated values in the next free space [3].

3.4.1 Data Lifetime in Data Files

After describing the process of how the database system handles insertion, deletion, and update of records in data files, we can look towards the lifetime of that data, particularly the deleted records that are still recoverable. Once an active record has been deleted and turned into an expired record, it will continue to exist as slack data. This can either be database slack or file system slack. Referring back to Figure 3, we see that in the first state (1), *t5* is an active record. In the second state (2) *t5* is deleted and becomes database slack. But once the vacuum process is executed, *t5* becomes file system slack in the fourth state (4) as the size of the data file was decreased [3].

Figure 4 gives a good illustration of the flow of data during its lifetime and possible transitions from being an active record to the desired state of the data being completely removed from disk. Following the lifetime of the record *t5* from Figure 3 with reference to Figure 4, we end up with *t5* being in the current state of being file system slack. For the data to be completely removed from the file system, the OS will have to execute its process for defragmenting the file system or use the data block for other data [3].

Distinguishing between database slack and file system slack is important and needs consideration when analyzing a system looking for deleted data. Database slack exists within the space allocated to the database system and therefore cannot be deleted by the OS processes for defragmentation or secure deletion [3]. File system slack is no longer within the space used by the database and therefore can be used by the OS to reallocate to another process or file. Someone with access to the file system will have access to any expired data that hasn't been fully removed in the database and file system slack space (as well as the active data still in use) [3].

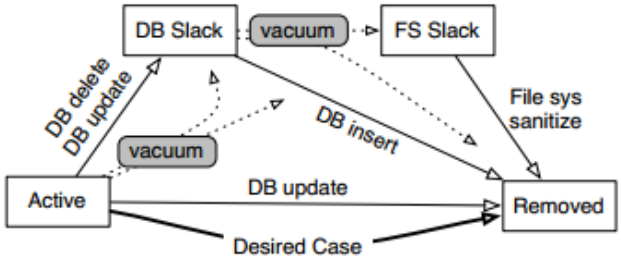

Figure 4. The flow of data during its lifetime [3]

4. INDEXES

Indexes are a popularly used structure to help reduce disk I/O and improve querying performance on database tables that contain a large set of data. A table without indexes will have to do a full table scan to find a specific value or record. Indexes also can retrieve a small set of randomly distributed rows from the table [9]. Typically an index is created for a specific column within a table. This column is typically involved in queries frequently such as in reports or aggregation queries or used as unique constraints on data in the table (such as ID). Indexes are objects similar to tables and are stored within the data files for the associated table space. Indexes are logically and physically independent from the

data that they are associated with. Therefore the creation and deletion of these indexes do not affect the actual structure containing the data [9].

4.1 Index Structure

The Oracle database system offers different types of indexes, such as B-tree indexes, Bitmap and Bitmap join indexes, function-based indexes, and application domain indexes. We will focus on B-tree indexes as these are the most common form of indexes used in database administration.

A B-tree index is an ordered list of values divided into ranges. B-trees are useful for improving the speed of queries that use and exact or range search [9]. The B-tree index structure is formed by two different types of blocks: branch blocks and leaf blocks. The branch blocks are used for searching and the leaf blocks store values. Therefore the branch blocks' ordered lists contain the ranges which point to each individual leaf block that contains the associated values. The leaf blocks contain the indexed data value and the corresponding rowid that points to the actual row in the table. Leaf blocks are also linked to their left and right sibling leaf blocks. It is important to note that indexes based on character values use the binary value of the characters based on the character set used by the database [9].

4.2 Index Analysis

Indexes are stored in the same tablespace and data files as the tables they are associated with. Therefore they also follow the same behavior regarding deletion and insertion of records (i.e creating database slack space). Indexes are automatically updated and maintained to reflect changes to data in the table from insertion or deletion operations [9]. However, the deleted entries from the index are not automatically removed from the space. Therefore, indexes have to be rebuilt to clean out any entries that are expired. This leaves information that can be recovered and help reveal the history of operations executed on the table.

When a record is deleted or updated (updates consist of a delete and insert) in a table, the index will contain a dead leaf node for that original record. This dead leaf node will continue to exist until the index is rebuilt or the space is reused, resulting in the node to be overwritten. But even with the rebuilding of the index, sometimes deleted node entries can persist in database slack space, similar to the discussed issue with data files in section 3.4.1 [3].

We can start analyzing an index by first validating the structure. Oracle provides a useful statement in their SQL language for validating and analyzing data structures within the database, including indexes [10]. To determine the current state of the index, we use the ANALYZE statement with the VALIDATE STRUCTURE option. Here we will analyze an index called employees_idx:

```
SQL> ANALYZE INDEX employees_idx VALIDATE STRUCTURE;
```

Should the statement return an error, then the index is corrupted, otherwise it is valid. This statement validated the index and populated the INDEX_STATS table in the SYS schema of the database. We can now query this table to view the statistics of the index.

```
SQL> SELECT name, height, lf_rows, lf_blks, del_lf_rows, distinct_keys, used_space FROM SYS.INDEX_STATS;
```

This query will return a result set similar to the results shown in Table 2. Note these are figurative numbers, not actual results from the above query.

Table 2. Current statistics of index from querying INDEX_STATS table

NAME	employees_idx
HEIGHT	2
LF_ROWS	235
LF_BLKS	562
DEL_LF_ROWS	74
USED_SPACE	1487

The result from the query shows that the index B-tree structure has 74 deleted leaf node entries. This could be interpreted as the index hasn't been rebuilt recently and when comparing the number of deleted leaf rows to the total number of leaf entries, there has been quite a bit of activity going on with the associated table, particularly updates and deletes.

With this analysis of the current state of the index structure, we can assume that the lifetime of deleted entries is relatively short considering that the system is better at reusing space in the B-tree structure compared to data files. However there are still other factors that could cause the existence of deleted entries to remain in database slack space such as the balancing of the B-tree or merging of leaf nodes causing the deleted entry to persist in the internal nodes of the structure [3].

5. RECOVERING DELETED DATA

Recovering deleted data from a database is an essential piece of a thorough analysis of a database system. Deleted data can be caused by a company hiding fraudulent information or after a successful attack on the system. An intruder will usually attempt to hide any trace of activity they had on the system by dropping objects they created such as tables or procedures [11]. But as we discussed in section 3 regarding data files, pieces of data can remain in the database and file system slack space that can be recovered and analyzed by a forensic examiner. These pieces of data can be put back together to help understand the actions the intruder took. We will also look at recovering data by analyzing undo segments and transaction logs.

5.1 Data Blocks

In Section 3.4 we discussed the structure of the data files. These data files are divided into data blocks[1], the size being determined by the database's DB_BLOCK_SIZE initialization parameter. There are different block types depending on the type of data the

[1] To clarify the structure of data files, data files contain segments, each containing data for a single structure. These segments are made of multiple extents storing specific types of data. These extents are made of multiple data blocks, which are at the finest level of granularity corresponding to a specific number of bytes on the physical disk [12].

block is storing, such as table data or index data [11]. Data blocks are a good place to start searching for possible deleted data and dropped objects seeing that all object data is stored in these data blocks. Figure 5 illustrates the structure of an Oracle data block.

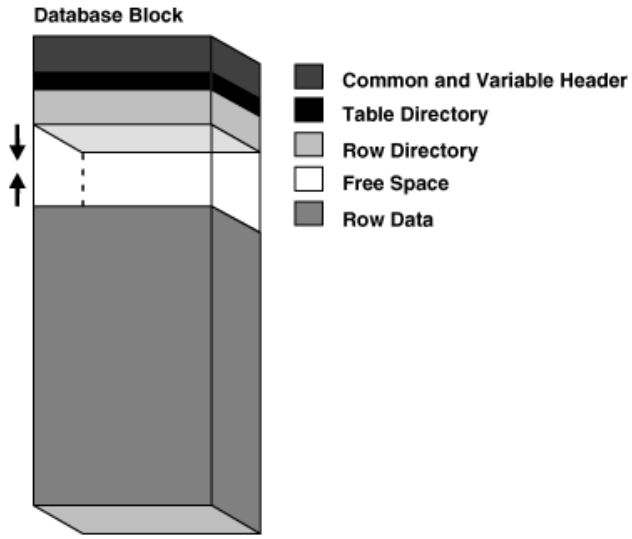

Figure 5. Data Block Format [12]

The block header contains information about the block, such as block type, Object ID of the structure it's assigned to (table, index, etc.), block address, and a checksum [11, 12]. The table directory contains information about the structure having rows in the block [12]. The row directory contains information about the row data in the block and address pointers to each actual data of the row [11, 12]. An important note about the row directory is that the space is not reclaimed when a row is deleted, but is reused when new rows are inserted into the block [12]. The row data is the actual data from a table or index and can span across multiple data blocks [12]. As new rows are inserted, entries are added to the row directory and free space is taken by the new row data entry. Once a block has been filled, the database system will start filling a new block, similar to an operating system's file system [11]. Each row of data has its own header of 3-4 bytes, depending on the size of the row. The first byte indicates the current status of the row using a set of flags [11]. The second byte determines lock status and the third byte is used for determining the amount of data in the row. If the amount is greater than 255 bytes then the row header is extended to 4 bytes allowing for the amount to be up to 65536 bytes [11]. Following the row header is the actual data.

5.1.1 Analyzing Data Blocks

When an entry in row data is deleted, it is marked in the header of the row as deleted in the first byte of the header. The fifth bit of the first byte is the "deleted" flag. A common set of flags for a row is 0x2C, which will become 0x3C once the "deleted" flag bit is set [11]. The row remains in the block until the space is reused for a new entry. Until the space is overwritten, the data remains easily recoverable since there is a link in the row directory pointing to the data. In some situations there will be an entry in row data without an entry in the row directory.

We can start analyzing a row entry by analyzing the header first. Figure 6 shows a sample hex dump of a single entry in row data.

```
189d3790h:                                                 3C 01 11
189d37a0h: 04 C3 06 13 2F 04 C3 06 13 2F 02 C1 37 0D 4D 59
189d37b0h: 5F 54 45 4D 50 5F 54 41 42 4C 45 02 C1 02 FF 02
189d37c0h: C1 03 07 78 6B 03 17 12 08 38 07 78 6B 03 17 12
189d37d0h: 08 38 07 78 6B 03 17 12 08 38 02 C1 02 FF FF 01
189d37e0h: 80 FF 02 C1 07 02 C1 02
```

Figure 6. Sample Hex Dump of Sample Row Data [11]

Looking at the row header (3C 01 11), we see immediately that the entry is marked as deleted since the first byte is set as 0x3C. The third byte shows that this row contains 0x11 columns (17 in decimal). After that is the data for all of the columns in the row. Each column within the entry is preceded with a size byte which we will use to count the number of bytes used in the column, leaving the next byte as the size byte for the next column [11]. We continue to do this until we reach the total number of columns indicated in the header. Following this method we reach the ending structure of the row as displayed in Table 3, including the size byte as the first byte followed by data for the column. Note that a column containing 0xFF indicates that there is no data present in the column.

Table 3. Breakdown of columns in sample row data entry [11]

Col 1	04 C3 06 13 2F
Col 2	04 C3 06 13 2F
Col 3	02 C1 37
Col 4	0D 4D 59 54 45 4D 50 5F 54 41 42 4C 45
Col 5	02 C1 02
Col 6	FF
Col 7	02 C1 03
Col 8	07 78 6B 03 17 12 08 38
Col 9	07 78 6B 03 17 12 08 38
Col 10	07 78 6B 03 17 12 08 38
Col 11	02 C1 02
Col 12	FF
Col 13	FF
Col 14	01 80
Col 15	FF
Col 16	02 C1 07
Col 17	02 C1 02

5.1.1.1 Locating Deleted Rows

When looking for deleted data, a good start would be the row directory of a data block. Follow each entry's pointer to the corresponding row in the row data section of the block and check the first byte of the row header. If the fifth bit is set, then the row has been marked as deleted. You can cycle through the row, blocking out the data just as we did in Table 3. Any data left that has not been blocked out is likely either deleted data or left over from an UPDATE operation to the data [11]. You can repeat this process for each block in the data file, which can be tedious. A better strategy is to look at each block that is assigned to a specific table or object of interest. You can determine blocks assigned to objects of interest simply by checking the header of the block for

the Object ID and comparing it to the ID of the object you are interested in examining.

It is important to note that when an object is created, a row is inserted in the object's table. If the table has any indexes, then entries are created for those indexes as well. And depending on the type of object, rows may be inserted in other tables as well. Therefore there may be multiple instances of the row data in different data blocks. When the object is dropped, all corresponding entries will be marked as deleted as well. Though the space can be reused by a new entry, the fact that there were multiple locations where data for the object was stored gives a better chance that not all of the corresponding deleted rows for the dropped object were reused when an examiner starts to investigate a system. Understanding this concept can help us determine locations in which to look for data of an object of interest [11].

5.2 Undo Tablespaces and Segments

Recent versions of the Oracle database system provides automatic undo management. That is, the system automatically configures and manages space and undo segment information to reverse changes made to the database [12]. The system provides a separate tablespace for undo segments to be stored. An image of the data is recorded and stored in an undo segment in the undo tablespace before a DML[2] operation is performed. This creates a convenient way of rolling back previous transactions.

5.2.1 Analyzing Undo Segments

Recall that segments consist of extents that make up individual specific data. Analyzing undo segments can give an idea of very recent operations that have been executed[3]. We can start by looking at the undo segment header. One thing of interest that the header contains is an extent map that holds the data block addresses that point to the first block of the extents and the lengths of each extent. Each individual extent has its own header that contains a set of offsets that point to the individual undo entries. To find the individual undo entry, it is the offset added to the block base plus twenty [13].

An undo entry corresponds to a specific action performed on the system, the most common being row and index operations. Each undo entry has a header containing information such as the type of operation the entry is for and the object ID the entry pertains to. Analyzing the undo entries can lead to evidence of what an intruder did if their actions required a type of transaction [13].

5.2.2 Undo Retention Period

Once a transaction has been committed, the undo data is no longer needed. There is however an "undo retention period", which is a time period that the database tries to keep old undo information before overwriting it. Once the information in an undo segment has become older than the retention period, it is marked as "expired" and the space is available to be overwritten [14]. This makes the expected lifetime of undo segments relatively short.

5.2.3 Using Undo Entries with ROLLBACK

Oracle provides a SQL statement for undoing changes using the undo segments stored in the undo tablespaces [16]. The ROLLBACK statement will undo any changes made in the current uncommitted transaction, changing the data back to the image of the data stored in the undo entries. Recall from Section 2.1 that transactions are not automatically written to the transaction logs until a COMMIT is executed or one of the other causes for the redo buffer to be emptied. Though most likely by the time an investigation has begun on the system, the transaction will have already been committed. It is however possible to roll back previously committed transactions using the FORCE clause along with the global transaction ID of the transaction you wish to roll back. The global ID can be found by querying the data dictionary view DBA_2PC_PENDING. For example if we wished to roll back the transaction with global ID 54, we would execute the following statement:

```
SQL> ROLLBACK FORCE '54';
```

Note that in order to execute this statement the user must have the FORCE TRANSACTION or FORCE ANY TRANSACTION system privilege depending on the user that committed the transaction [16].

5.3 Transaction Logs

In Section 2 we discussed transaction logs, what information they contain of forensic interest, and did a general analysis of the transaction logs. For a forensic investigation of a database system you may need to examine particular DML or DDL operations within the transaction logs' redo entries to discover the activity of an intruder of the system and possibly recover any data that was deleted since DELETE transactions will show up in the redo logs.

5.3.1 Example Examination of DML operations in Redo Entries

Figure 7 is a sample hex dump of a redo entry that contains an INSERT operation being performed [7].

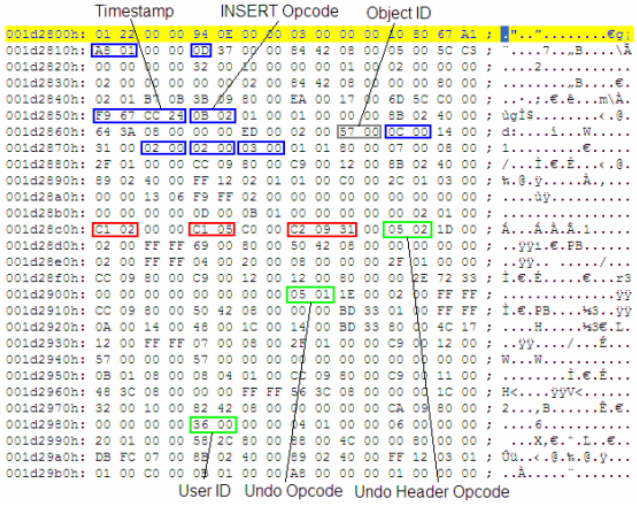

Figure 7. Sample hex dump of a redo entry performing an INSERT operation [7]

[2] DML stands for "Data Manipulation Language" and pertains to statements that alter data within a structure such as INSERT, UPDATE, or DELETE statements.

[3] It is important to understand the difference between undo segments and transaction (redo) logs since the two concepts seem similar. Undo segments contain information about how to undo changes made to the data in the database. Transaction (redo) logs contain information on how to reproduce a change made to data.

Analyzing the dump, we see that the opcode for the change vector is 0x0B02 (11.2) which according to Table 1is an INSERT on a single row. Preceding the opcode is the timestamp for the entry and 22 bytes past the opcode is the object ID for the table that INSERT was executed on. An examiner of the system can then determine the owner of the object and the name of the object in question. Further analysis of the hex dump will show that three columns were inserted: the 2 bytes inserted in the first and second columns and 3 bytes inserted into the third column. Following the array of entries there is "op" and "ver" and depending on the number entries, their location will either be 1 (if even) or 2 (if odd) bytes after the array of entries [7]. In our example in Figure 7, our array of entries has 6 entries: 3 for the columns inserted and 3 for dealing with the size of the array (0x0C) and semi-fixed values (0x0014 and 0x0031) [7]. Therefore "op" is immediately after our array. The location of the actual data for the inserted columns varies depending on the value of "op", which is 1, 2, or 17 [7]. Table 4 lists the locations of the column data based on the value of "op".

Table 4. Locations of column data based on value of "op" [7]

Value of "op"	Location of data in number of bytes from "op" location
1	72
2	64
17	120

In our example "op" is 1, therefore the data for the columns starts at 72 bytes from the "op" location. This is highlighted in red in Figure 7 [7]. Now we know the values that were inserted into the columns of the object.

6. DATA AT REST ENCRYPTION

Data at rest encryption, such as Oracle's Transparent Data Encryption (TDE) gives the ability to encrypt and decrypt sensitive data stored in the database on the fly for any users or applications that have access to the data. This concept helps protect data stored on disk in the event of the data files or hard disk being stolen [17].

TDE can be used to either encrypt specific columns in a table or entire tablespaces. Oracle stores the encryption keys used to decrypt the data in a security module external to the database system, typically as an Oracle wallet or Hardware Security Module (HSM) [17]. The encryption key is used to decrypt the table or tablespace key which is used to encrypt and decrypt the data within the table or tablespace, giving a two-level encryption architecture [17].

6.1 Forensic Importance of TDE

While TDE offers good benefits of security to system administrators, it offers a challenge to forensic examiners in certain situations. In the case of TDE encrypting entire tablespaces, the data files containing all of the data are completely encrypted. Therefore a physical backup of the data files also requires a copy of the master encryption key which is either stored in a wallet file or HSM. If we take an example of the database system being on a damaged disk, therefore inoperable, recovering the encrypted data files and encrypted wallet file can be a tedious task. Determining what data on the disk is the encrypted data files and finding the data for the wallet file storing the encryption can be very challenging. In the case of the master encryption key being stored on a HSM, taking a physical copy of the data files will also require you disassemble and take the HSM out of the target machine. This method may conflict with the laws and regulations of obtaining digital evidence and cause any evidence on the system to be invalid for use in a court case.

7. DATA TAMPERING

Data tampering can occur when an intruder gains access to a DBMS. Some database systems have built-in mechanisms to help detect tampering, such as audit log security. Audit log security is a component that tracks documents and their versions, ensuring previous versions of a document cannot be altered [1]. But what if an intruder accessed the system and modified these logs, essentially hiding traces of their activities on the system? A forensic examiner or system administrator will need to know techniques and methods for determining if the audit logs are correct and unaltered. In the section we discuss concepts and methods for detecting data tampering.

7.1 Detection with Hash Functions and Notarization

Pavlou and Snodgrass proposed a concept for detecting data tampering that revolves around using a strong one-way hash function to prevent data tampering [1]. This is done by hashing the data changed by a transaction and at a later period of time validating the data by rehashing it and comparing it with the original hash value that is store in an external location. If any data tampering has occurred, it will appear during the validation step when the hash values do not match [1]. Pavlou and Snodgrass describe these two phases as *online processing* and *validation*. During online processing, the data is hashed along with a timestamp and the value is sent to an external notary service that stores a copy of the value and returns a notary ID. During validation, a validation service rehashes the data and sends the new hash value to the notary service along with the original notary ID. The notary service compares the new hash with hash value copy it was previously sent to determine if they match. Figure 8 illustrates the processes of online processing and validation described by Pavlou and Snodgrass [1]. If an intruder gains access to the system and manipulates any data, the hash values of the data will not match. This concept however is assuming that the notarization and validation services are stored in a secure and trusted location.

7.2 Detection by Hashing of Audit Logs

Another concept proposed by Azemović and Mušić is a data tamper detection model focused on collecting and securing validity of collected data [4]. Their model is as follows:

1. A user modifies data within the database [4]
2. Activity information is logged in a data warehouse [4]
3. Validation components secure collected audit logs [4]

In more detail, normal operation is performed by a user on the system (1). The transaction passes through a detection layer comprised of triggers that are defined by a specific event and are executed when that event occurs (such as insertion of a new record) (2). Then the audit logs are validated and stored. Newly inserted audit logs pass through another layer of security that

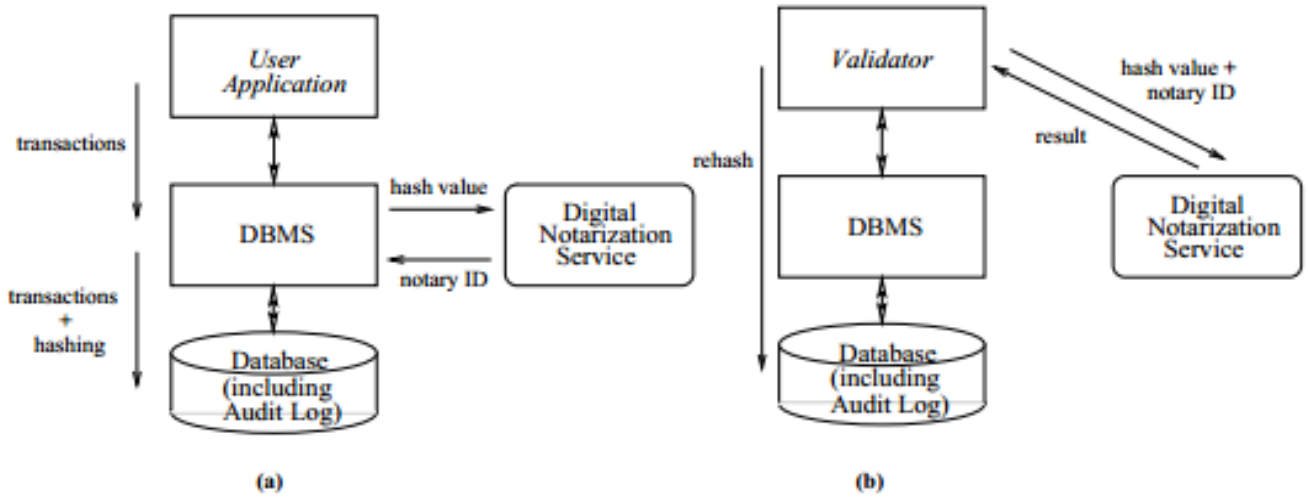

Figure 8. Online Processing (a) and Audit log validation (b) [1]

involves hashing two different hash values of the audit log row and storing the values in the record with the log information [4]. The two special hash columns added are HReserved (horizontal) and VReserved (vertical).

The horizontal hash is computed by hashing all of the column values of the row except for the HReserved and VReserved columns. Any tampering of the audit log data will cause the rehash of the row to mismatch the hash value stored in HReserved [4]. The VReserved is computed using the HReserved hash values of the last two rows and the current row. This creates a chain like hash among the audit log rows. This will detect data tampering if a particular row is deleted from the audit log table [4].

8. LIVE RESPONSE

In the event of an intrusion on a database system, a common reaction for a system administrator is to disconnect the database from the network or shutdown the system to prevent any further intrusions or data theft. This however could cause useful evidence stored in volatile storage such as memory to be lost, making an investigation of the system slightly more difficult. Due to these types of issues, it is preferred the analysis on the system while it's still running and connected to the network just as it would normally be running. This is referred to as a "Live Response" [14].

Live Response is the process of collecting volatile data that would essentially be lost if the system was powered down for an offline analysis. It also gives the examiner a chance to collect information from any human-friendly sources such as event logs [14]. It is best to perform a Live Response analysis with the presence of a system administrator that is familiar with the system and its configurations. This person can also act as a witness in a later court case, justifying procedures taken by the examiner. Also this person can help spot any suspicious findings such as a new user being created [14].

There are some issues with Live Response though. Since a Live Response analysis is performed on a compromised system that is running, the trustworthiness of the information that can be found can be questioned. Since it is a compromised system, the attacker could have modified other files and programs within the operating system, possibly causing the operating system to not act as expected [14]. Also even though the main benefit of a Live Response analysis is gathering information from volatile memory, it is also an issue. There is no way to reproduce a Live Response analysis because once the volatile data is gone, there's no way to recover it [14]. This could cause evidence to be easily challengeable in court.

8.1 Live Response Steps

First it should be understood that during a Live Response analysis it is impossible for the examiner to not leave a footprint [14]. That said, the examiner must still take extreme caution when dealing with the system. There are some things that unavoidable such as logging on the system which will likely be logged somewhere, but such things are considered acceptable [14]. However the examiner must be careful to not write any files to the system as this could overwrite deleted data that could have been recovered during an offline analysis.

General steps to follow during a Live Response analysis include [14]:

- Record system date and time
- Collect a list of users currently logged on the system
- Collect a list of all users and groups and information such as last login and group membership
- Collect a list of open ports and connections
- Collect a list of current processes running
- Collect a list of libraries or shared object files that are loaded by each process
- Perform memory dumps of all running processes and a full system memory dump
- Get file names, MAC times, and file attributes of entire disk
- On a Windows system, dump registry information
- Collect copies of log files and message logs

These general steps can apply to any system and can be applied to any live analysis of a system in general. When looking for Oracle specific items, determining the location of many of the database

system files (log files, trace files, and control files) can be scattered throughout the system. A good way to start locating Oracle specific items is to check the system environment variables for the ORACLE_HOME location. Once the location of Oracle home is determined, the examiner should grab the startup parameter file which contains information about log and trace file locations [14]. It is important to note that what is listed in the startup parameter file may not be want the database system is currently using since the attacker could have used an "ALTER SYSTEM" statement [14].

Oracle specific items of interest are as follows [14]:

- Audit files location
- Background processes trace files and alert.log file
- Oracle server core dumps
- Archived redo logs in the flash recovery area
- Trace information for user processes
- Control files
- Data files
- External files written from Oracle
- Listener log files

Once the examiner has collected all items of interest listed, the final thing to do is log in to the database using SQLPlus. This is the last step because logging in will create log entries and change rows in system tables such as V$SQL [14]. The examiner should take caution to not perform any DML or DDL operations on the system once logged in such as DELETE statements or DROP statements. The examiner should try to log in with SYS privileges if possible which will allow them to access information needed for analysis [14].

Once logged in the database system, the very first bit of information the examiner should grab is a list of recently executed SQL statements [14]. This gives a possibility of finding useful evidence of statements executed by the attacker. Once that is finished, the examiner can start gathering information pertaining to database objects. A list of information of interest within the database system is as follows [14]:

- Logon information
- List of users and user roles and system privileges
- List of all objects (ID, owner, name, and type)
- List of dropped objects
- List of block changes
- Server version and parameters
- List of external files
- List of all data files
- Trigger information
- Library information
- List of database links
- Database job information
- PL/SQL object information
- Java objects information and source

Once the examiner has gathered all useful information to analyze for forensic evidence, the database server can be shut down and taken off the network assuming that is what the owner requests [14]. The examiner can now take the collected information and analyze it for evidence of the attacker's actions and determine how the attacker gained access to the system and what exactly they did.

9. REFERENCES

[1] Pavlou, Kyriacos E. and Snodgrass, Richard T. 2008. Forensic Analysis of Database Tampering. ACM Transactions on Database Systems, Vol. V, No. N, September 2008, Pages 1-45 http://www.cs.arizona.edu/people/rts/pubs/TODS08.pdf

[2] Guimaraes, Mario A.M., Austin, Richard, and Said, Huwida. 2010. Database Forensics. InfoSecCD '10 2010 Information Security Curriculum Development Conference, Pages 62-65 http://dl.acm.org/citation.cfm?id=1940958

[3] Stahlberg, P., Gerome, M., and Levine, B.N. Threats to Privacy in the Forensic Analysis of Database Systems. University of Massachusetts http://forensics.umass.edu/pubs/stahlberg07forensicDB.pdf

[4] Azemović, Jasmin., Mušić, Denis. 2009. Efficient Model for Detection Data and Data Scheme Tempering with Purpose of Valid Forensic Analysis. IPCSIT, Vol. 2 (2011) http://www.ipcsit.net/vol2/16-A196.pdf

[5] Oracle Database Administrator's Guide: What is the Redo Log? http://docs.oracle.com/cd/B28359_01/server.111/b28310/onlineredo001.htm

[6] Oracle Database Utilities: Using LogMiner to Analyze Redo Log Files http://docs.oracle.com/cd/B28359_01/server.111/b28319/logminer.htm

[7] Litchfield, David. 2007. Oracle Forensics Part 1: Dissecting the Redo Logs. NISR Publication http://www.dcs.co.jp/security/NGS_freedownloads/OracleForensicsPart1_Dissecting_theRedoLogs.pdf

[8] Oracle Database Concepts: Physical Storage Structures http://docs.oracle.com/cd/E11882_01/server.112/e25789/physical.htm

[9] Oracle Database Concepts: Indexes and Index-Organized Tables http://docs.oracle.com/cd/E11882_01/server.112/e25789/indexiot.htm

[10] Oracle Database Administrator's Guide: Analyzing Tables, Indexes, and Clusters http://docs.oracle.com/cd/B28359_01/server.111/b28310/general002.htm

[11] Litchfield, David. 2007. Oracle Forensics Part 2: Locating dropped objects. NISR Publication http://www.databasesecurity.com/dbsec/Locating-Dropped-Objects.pdf

[12] Oracle Database Concepts: Data Blocks, Extents, and Segments http://docs.oracle.com/cd/B28359_01/server.111/b28318/logical.htm

[13] Litchfield, David. 2007 Oracle Forensics Part 6: Examining Undo Segments, Flashback, and the Oracle Recycle Bin. NISR Publication http://www.databasesecurity.com/dbsec/oracle-forensics-6.pdf

[14] Litchfield, David. 2007. Oracle Forensics Part 4: Live Response. NISR Publication http://www.bandwidthco.com/whitepapers/database/Oracle%20Forensics%20Part%204%20-%20Live%20Response.pdf

[15] Oracle Database Administrator's Guide: Introduction to Automatic Undo Management http://docs.oracle.com/cd/B28359_01/server.111/b28310/undo002.htm

[16] Oracle Database SQL Language Reference: ROLLBACK http://docs.oracle.com/cd/B28359_01/server.111/b28286/statements_9021.htm

[17] Oracle Database Advanced Security Administrator's Guide: Securing Stored Data Using Transparent Data Encryption http://docs.oracle.com/cd/E29505_01/network.1111/e10746/asotrans.htm

10. ABOUT THE AUTHOR

Robert Horn is a student studying Software Engineering at Auburn University and a part-time software developer at Health Information Designs, Inc. in Auburn, Alabama with experience in web application development, Oracle database administration and reporting, and database quality assurance.