DOI: http://dx.doi.org/10.5769/C2011022

Finding Detached Microsoft SQL Server Database Files

Fábio Caús Sícoli[a], André Morum de Lima Simão[b]

Abstract — During the execution of a search warrant, one may try to hamper law enforcement officials by hiding database artifacts. One way this can be done is by detaching a given database, which will drop all its metadata and make it invisible to the DBMS. This paper describes Microsoft SQL Server's database files and presents an algorithm capable of finding and extracting metadata from those files still present in the file system, in order to be scrutinized by forensics teams.

"Keywords — database; forensics; anti-forensics; MS SQL Server; detachment"

1. Introduction

People and organizations use different kinds of repositories to store their data. Around 92% of data is stored digitally [1], in medias such as text documents, spreadsheets, e-mails, web pages and databases. The latter has a few advantages, for example, making information available to simultaneous users and applications, ease of access and manipulation of data [2]. In this regard, organizations are increasingly using Database Management Systems (DBMSes) to store digital information in a structured fashion [3].

Databases store information related to the activities carried out by organizations. They may contain financial data, accounting information, as well as client lists and product portfolios. As a consequence, in order to protect sensible information regarding the operation of organizations, there might be mechanisms to hide either the DBMS itself or its stored data. Among the most popular DBMSes are the Oracle RDBMS, Microsoft SQL Server, IBM DB2, MySQL and PostgreSQL [4].

During the execution of search warrants, law enforcement agencies look for evidence related to an alleged criminal act under investigation [5]. Through the analysis of information contained in a database, illegal activities may be verified, for instance, money laundering, social security frauds and tax evasion. As a result, anti-forensic procedures may be performed with the purpose of hiding database artifacts. A few examples are: unplugging the DBMS server from the network, stopping the DBMS service and detaching a specific database. In this last case, the database will be invisible to the management system and its metadata will be erased.

To solve the problem of finding database files concealed by means of detachment, Microsoft SQL Server database files were studied in depth, using reverse engineering techniques, in order to identify their physical structures and eventually present an algorithm capable of identifying files belonging to detached databases.

The rest of this paper is organized as follows: Section 2 presents the storage structures of a Microsoft SQL Server database. Section 3 discusses the process of detaching a database. In Section 4, algorithms by other authors are studied, a new algorithm is proposed and test results are displayed. At last, Section 5 presents the conclusions of the present work.

2. Microsoft SQL Server database files

Microsoft SQL Server makes use of two kinds of files to map a database: log files and data files [6]. A database must have at least one file of each kind. On the other hand, a given file can not belong to more than one database [7].

Log files keep historical data of a database's transactions and are used for data recovery purposes [8]. The default extension for this kind of file is .LDF (Log Data File). Conceptually, a log file is a set of log records. Physically, the sequence of log records is stored in a set of files that constitute the transaction log. The SQL Server Database Engine internally splits each log file in virtual log files. They have no fixed size and their quantity is variable in a given physical log file.

Data files store objects, including tables, records, indexes, procedures and views. These files are classified into two categories:

- Primary: it contains the database's startup information and references to other database files, if they exist. It is recommended that the extension .MDF (Master Data File) is used to name primary files;
- Secondary: it holds data that exceed the storage capacity of a primary file, complementing them. Their default extension is .NDF (Not-a-master Data File).

2.1. Data files physical structure

Microsoft SQL Server data files are divided into pages. They are the fundamental data storage unit and the minimal structure that may be physically read or written by the SQL Server [9].

Up to its version 6.5, pages were 2KB large [10]. However, since release 7.0, page size was increased to 8KB. Pages are grouped into extents, composed by eight pages each.

At the beginning of each page, lies its header. It has 96 bytes to store information, such as its number, type and object to which it belongs. The data records area is located right after the header. It may also contain free space. Next, there is a table that keeps the location of records stored within that page. Figure 1 below displays the structure of a data file page.

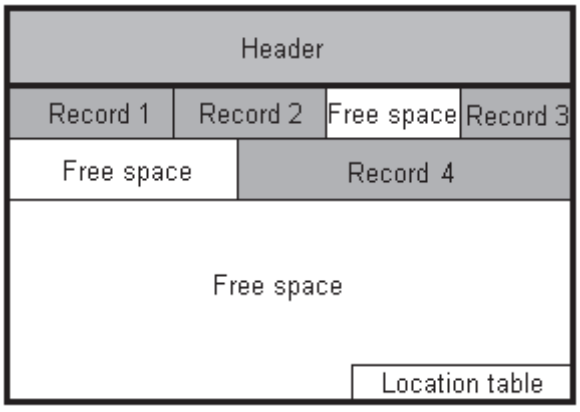

Figure 1. Page structure of a data file page

The header of a page keeps its main information divided into fields [11]. For the purpose of this paper, the most relevant fields are the "m_pageId" and the "m_type", described as follows:

- m_pageId: it contains a unique number that identifies the page within a data file. It has 4 bytes and is located at 32 bytes from the beginning of the page.
- m_type: this indicates the page type. It is the second byte of the page and its possible values are listed in Table 1.

Table 1. M_type values and their description

Value	Page Type	Description
1	Data page	Stores records from a heap table or from the last level of a clustered index.
2	Index page	Holds index records from the superior levels of a clustered index or from all levels of a non-clustered index.
3	Text mix page	Contains pieces of large objects and portions of text trees.
4	Text tree page	Keeps segments from a given attribute of a large object.
7	Sort page	Stores temporary results from a sort operation.
8	GAM (Global Allocation Map) page	Holds information about the allocation of extents. The third page of a data file is the first GAM page.
9	SGAM (Shared Global Allocation Map) page	Contains mixed extents allocation information. The first SGAM page of a data file is the file's fourth.
10	IAM (Index Allocation Map) page	Keeps allocation information related to indexes.
11	PFS (Page Free Space) page	Stores allocation and free space information. The first page of this kind is the second page of a file.
13	Boot page	Holds general information about a database. There is only one page of this type in each database.
15	File header page	This is the first page of the file. It contains information related to a file, such as its unique identifier, group of files to which it is associated, size and growth rules.
16	Difference map page	Stores information regarding which extents have been modified since the last differential or full backup. The seventh page is the first difference map page of a file.
17	ML map page	Keeps track of extents that were modified by minimally-logged operations since the last full backup. The first page of this kind is the eighth page of a given file.

2.1.1. Pages with database's metadata

The tenth page of every database's primary data file is a boot page [12]. It contains information about the base itself [13], such as the attributes described in Table 2.

Table 2. Attributes in a boot page

Attribute	Description	Location within the page
dbi_version	Version number of the current DBMS.	From position 100 to 101.
dbi_ createVersion	Version number of the DBMS where the database was initially created.	From position 102 to 103.
dbi_crdate	Database creation date and time.	From position 140 to 143 (time, in time ticks after midnight) and from position 144 to 147 (date, in days after 01/01/1900).
dbi_dbname	Name of the database.	From position 148 to 403.
dbi_dbid	Unique identification number.	Position 408.

Another page worthy of notice is the file listing page. It is the thirty-third of a primary data file and holds records that list every data and log file that make up the database. Each re-

cord related to a given file has 792 bytes and contains the following attributes: file_id, logical name and full path, located at the positions 8, 10 and 265, respectively, from the start of the record.

3. Database detachment

The procedure of detaching a database is generally carried out when one wants to move its data and log files to a different physical location or to another server instance [14]. In order to do that, one may use the stored procedure "sp_detach_db" or use the "SQL Server Management Studio Object Explorer" graphic user interface.

When a database is detached, it is removed from its SQL Server instance. This way, the base may no longer be accessed or seen by the DBMS. In addition, the base's metadata in the administrative tables "master.dbo.sysdatabases" and "master.dbo.sysaltfiles" are erased. However, the base itself remains intact, along with its data files and log files. Therefore, it can be reattached to the same instance where it was or to any other by using those files.

4. Methodology

The goal of the present work was to identify files from a detached Microsoft SQL Server database. At first, existing algorithms that claimed to resolve that issue were tested and their limitations were assessed. Then, physical structures of data and log files were examined, which led to the identification of their main attributes' contents and location. As a result, a new algorithm was proposed and run against a test environment.

In the following sections, existing algorithms are discussed, a new algorithm is presented and test results are displayed.

4.2. Existing algorithms

Paul Els [15] published an article with an algorithm to identify detached database files. He suggested to locate files named with the extensions MDF, NDF and LDF on disk and then query the "master.dbo.sysaltfiles" table to check which of those files were currently assigned to active databases. A file which was not associated to any database was pointed as being from a detached one.

However, Els' algorithm has some limitations. First of all, the algorithm examines only files having the default name extensions. Even though it is recommended to use default extensions to name database files, it is not mandatory. Therefore, there may be files that have those extensions but do not belong to databases, in addition to actual database files without default extensions in their names. In this regard, it is suggested that the algorithm analyzes files based on their contents, instead of on their names. Another downside is that the algorithm tries to make use of a connection to the DBMS, which may have been shutdown. Moreover, while the search is being carried out, one may not have proper credentials to be able to read the "sysaltfiles" table. Yet another scenario not addressed by Els' algorithm occurs when the volume that stores the files is offline. Finally, the algorithm uses the "xp_cmdshell" function, which is disabled by default from version 2005 onwards.

Ozar [16] also proposed an algorithm intended to find the same class of database files. He introduced a verification to test if the "xp_cmdshell" was enabled before trying to use it. Ozar's algorithm is similar to Els' [15], sharing the described limitations, such as the name extension based identification and the need to connect to the database service. It has another shortcoming of not being able to verify multiple DBMS instances.

4.3. Proposed algorithm

The algorithm assumes that files of a detached database will be readable by the operating system. On the other hand, files from an active base, belonging to a running DBMS instance, will not be available for reading or writing by processes other than the DBMS itself. This happens because the DBMS holds locks to its active files, not letting other processes access them. Thus, the algorithm is based on the assumption that the files it can actually read are not related to active databases.

The first step of the algorithm execution is to verify if the examined file has the leading hexadecimal sequence 01 0F 00 00. If so, the file has the fingerprint of a Microsoft SQL Server file. The first eight bytes of such a file keep respectively the following fields: m_headerVersion, m_type, m_typeFlagBits, m_level, m_flagBits, m_objId and m_indexId (two bytes) [17].

Then, the algorithm checks if the file has the characteristics of a data file. As such, it has a PFS, a GAM and an SGAM page as second, third and fourth page, respectively. That way, the "m_type" field values of those pages have to match the corresponding values of those kinds of pages. If that is the case, the examined file will be classified as a data file. Otherwise, as a log file.

Data files may be of the primary or secondary kind. To differentiate between them, their tenth page must be read. If it is a boot page, the file will be classified as a primary data file. If not, as a secondary data file.

After the file identification phase is done, additional information may be obtained. If the file is a primary data file, its creation date, its version and the version of the DBMS where the file was created may be read from its boot page. In addition, from the file listing page, the following attributes are obtained: file_id, logical name and full path of data and log files which make up the database. Furthermore, from a secondary data file, its logical name may be extracted.

4.4. Deployment of the algorithm in a test environment

The aforementioned algorithm was implemented as a program in the Perl language. Then, it was run on a disk partition containing 76 detached database files of Microsoft SQL Server

versions 7.0, 2000, 2005, 2008 and 2008 R2. The environment had also four files from active databases, which belonged to a running DBMS.

From the primary data files, the algorithm obtained the name of the database to which the file used to belong and a list of other data and log files from the same base. Besides, logical names were extracted from secondary data files. Results were compared with the ones obtained by means of the SQL Server's "DBCC checkprimaryfile" command [18]. The algorithm identified every file correctly and there were no false positives.

Additional tests were carried out in the same environment, but this time the DBMS was not running. All the 76 detached database files were correctly identified by the proposed algorithm. In this scenario, however, it produced false positives, indicating active database files as being from detached ones.

Finally, El's and Ozar's algorithms were run on the same test environments. Both yielded the same results. On the first environment, the algorithms recognized 54 out of 76 detached database files. They missed 22 files, all of which didn't have the default name extensions. On the second scenario, however, the algorithms could not be executed. Since they were written in the SQL Server scripting language, they require that the DBMS is running, which was not the case.

The proposed algorithm was considered successful. Although there were some false positives when the DBMS was not running, the main goal of the algorithm was achieved, not letting database files go unnoticed. In addition, it also performed better than the existing algorithms on both scenarios.

Changes to the proposed algorithm were considered to reduce false positives. Querying the "master.dbo.sysaltfiles" table to verify files currently assigned to databases would not be achievable for the very situation of the DBMS not being started, which itself produced the false positives during the tests. Alternatively, one could delve into the offline DBMS data files looking for the actual records on disk from the mentioned system table.

5. Conclusion

Among anti-forensic procedures that may be carried out on a database, detachment leads to erasing its metadata and making the base invisible to the DBMS.

Existing algorithms to find files from detached Microsoft SQL Server databases [15, 16] are ineffective in several situations, such as when database files do not have default name extensions, the DBMS is offline or administrative credentials are unavailable.

In order to create a new algorithm, physical structures of data and log files were scrutinized to bring forward their main attributes' contents and location.

As a consequence, an algorithm was conceived to find detached Microsoft SQL Server databases files by actually going into files' contents and interpreting their internal structures. Thereupon, it was implemented in the Perl programming language and it was run on a test environment with 76 files from detached databases. The proposed algorithm was able to correctly identify questioned files and extract their main metadata, being more effective than existing algorithms for the same purpose.

Acknowledgements

This paper was produced with institutional support by the Brazilian Federal Police (DPF) and with financial aid from the National Public Security and Citizenship Program (PRONASCI), an initiative led by the Ministry of Justice. The studies were carried out under the supervision of Professors from the Electrical Engineering Department at University of Brasilia, who contributed to directing the efforts and producing high level scientific knowledge.

References

[1] Peter Lyman and Hal R. Varian, How much information? (2003). Available at: <http://www2.sims.berkeley.edu/research/projects/how-much-info-2003/>. Accessed on 15/09/2010.

[2] Yvonne Miller, History of DBMS/Relational Database (2010). Available at: <http://juicyarticles.net/articles/History-of-DBMSRelational-Database-5356>. Accessed on 15/09/2010.

[3] Miranda Welch, Information governance and Stewardship for Records and Information Management (2009). Available at: <https://www.cis.unisa.edu.au/ wki/images/5/58/Welch_thesis_June_01_2009.doc>. Accessed on 17/09/2010.

[4] Colleen Graham et al, Market Share: RDBMS Software by Operating System, Worldwide (2009). Gartner 2009 Worldwide RDBMS Market Share Reports.

[5] Antonio Miranda de BARROS, Busca e apreensão no Processo Penal (2007). Available at: <http://sisnet.aduaneiras.com.br/lex/doutrinas/arquivos/060907.pdf>. Accessed on 19/09/2010.

[6] Buck Woody, SQL Server Reference Guide: Files and Filegroups (2009). Available at: <http://www.informit.com/guides/content.aspx?g=sqlserver&seqNum=42>. Accessed on 20/09/2010.

[7] MSDN Library, Understanding Files and Filegroups (2010). Available at: <http://msdn.microsoft.com/en-us/library/ms189563.aspx>. Accessed on 20/09/2010.

[8] MSDN Library, Files and Filegroups Architecture (2010). Available at: <http://msdn.microsoft.com/en-us/library/ms179316.aspx>. Accessed on 20/09/2010.

[9] MSDN Library, Understanding Pages and Extents (2010). Available at: <http://msdn.microsoft.com/en-us/library/ms190969.aspx>. Accessed on 20/09/2010.

[10] Tom Pullen, Data Page Structures in SQL Server 6.5 (2001). Available at: <http://www.sql-server-performance.com/articles/dba/65_data_structure_p1.aspx>. Accessed on 20/09/2010.

[11] Paul S. Randal, Inside the Storage Engine: Anatomy of a page (2007). Available at: <http://sqlskills.com/BLOGS/PAUL/post/Inside-the-Storage-Engine-Anatomy-of-a-page.aspx>. Accessed on 22/09/2010.

[12] SQL Server Simplified, Architecture Series 2 - Boot page (2008). Available at: <http://sqlsimplified.com/architecture_series_2>. Accessed on 22/09/2010.

[13] Paul S. Randal, Boot pages, and boot page corruption (2008). Available at: <http://www.sqlskills.com/blogs/paul/post/Search-Engine-QA-20-Boot-pages-and-boot-page-corruption.aspx>. Accessed on 22/09/2010.

[14] MICROSOFT Technet, Detaching and Attaching Databases (2010). Available at: <http://technet.microsoft.com/en-us/library/ms190794.aspx>. Accessed on 20/09/2010.

[15] Paul Els, Managing Free Space (2009). Available at: <http://www.sqlservercentral.com/articles/Administration/67692/>. Accessed on 23/09/2010.

[16] Brent Ozar, Find Detached Databases (2009). Available at: <http://sqlserverpedia.com/wiki/Find_Detached_Databases>. Accessed on 23/09/2010.
[17] Eka Siswanto, Unraveling MS SQL 2000 MDF Format (Part 1) (2008). Available at: <http://eka-siswanto.blogspot.com/2008/06/unraveling-ms-sql-2000-database-format.html>. Accessed on 22/09/2010.
[18] Alexander Chigrik, Useful Undocumented Maintenance SQL Server 2008 DBCC Commands (2009). Available at: <http://www.sswug.org/articles/viewarticle.aspx?id=46986>. Accessed on 24/09/2010.

Fábio Caús Sícoli has a bachelor degree in Computer Science from University of Brasilia (2004) and a postgraduate degree in Cryptography and Network Security from Fluminense Federal University (2010). He is also a masters student in Computer Forensics and Information Security in the Electrical Engineering Department at University of Brasilia. He has been working as a forensic expert in computer crimes in the Brazilian Federal Police for the last six years.

André Morum de Lima Simão has a bachelor degree in Computer Science from Brasilia Catholic University (2000) and a postgraduate degree in Information Security Management from University of Brasilia (2002). He joined the Brazilian Federal Police's forensic experts team six years ago, where he has been working since then. Nowadays, he is also a masters student in Computer Forensics and Information Security in the Electrical Engineering Department at University of Brasilia.