Digital Investigation 14 (2015) S106–S115

Contents lists available at ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin

DFRWS 2015 USA

Database forensic analysis through internal structure carving

James Wagner [a], Alexander Rasin [a, *], Jonathan Grier [b]

[a] DePaul University, Chicago, IL, USA
[b] Grier Forensics, USA

Keywords:
Database forensics
File carving
Memory analysis
Stochastic analysis
Database storage modeling

A B S T R A C T

Forensic tools assist analysts with recovery of both the data and system events, even from corrupted storage. These tools typically rely on "file carving" techniques to restore files after metadata loss by analyzing the remaining raw file content. A significant amount of sensitive data is stored and processed in relational databases thus creating the need for database forensic tools that will extend file carving solutions to the database realm. Raw database storage is partitioned into individual "pages" that cannot be read or presented to the analyst without the help of the database itself. Furthermore, by directly accessing raw database storage, we can reveal things that are normally hidden from database users.

There exists a number of database-specific tools developed for emergency database recovery, though not usually for forensic analysis of a database. In this paper, we present a universal tool that seamlessly supports many different databases, rebuilding table and other data content from any remaining storage fragments on disk or in memory. We define an approach for automatically (with minimal user intervention) reverse engineering storage in new databases, for detecting volatile data changes and discovering user action artifacts. Finally, we empirically verify our tool's ability to recover both deleted and partially corrupted data directly from the internal storage of different databases.

Introduction

Because most personal and company data is stored in digital form, forensic analysts are often tasked with restoring digital data contents or even reconstructing user actions based on system snapshots. The digital data recovery process is composed of both hardware and software phases. Hardware techniques extract data from physically damaged disks, while software techniques make sense of the recovered data fragments. Our work presented here focuses on software-based restoration techniques in the context of relational database management systems (DBMSes). A well-recognized forensic technique is the process of "file carving" that bypasses metadata and inspects file contents directly. If a sufficient proportion of the file can be recovered and recognized, then the content of the file (e.g., images or document text) can then be restored.

It is our contention that a significant amount of data, particularly what is referred to as *Big Data*, is not stored in flat files, but rather resides in a variety of databases within the organization or personal devices. Standard file carving techniques are insufficient to meaningfully recover the contents of a database; indeed, without the metadata of the DBMS (*catalog*), the contents of database tables could not be presented to the forensic analyst in a coherent form. The work presented here thus bridges this gap by introducing a **novel database carving approach** that allows us to reconstitute database contents and reason about actions performed by the database users.

* Corresponding author.
E-mail addresses: jwagne32@mail.depaul.edu (J. Wagner), arasin@cdm.depaul.edu (A. Rasin), jdgrier@grierforensics.com (J. Grier).

http://dx.doi.org/10.1016/j.diin.2015.05.013

Our contributions

We present a comprehensive collection of techniques for forensic analysis of both static and volatile content in a database:

- We define **generalized storage layout parameters** for parsing the raw storage (including the volatile kind) of many different relational databases.
- We compare and contrast **different storage design decisions** made by a variety of DBMSes and discuss the resulting implications for forensic analysis.
- We present a tool that can **reverse-engineer new DBMS storage parameters** by iteratively loading synthetic data, executing test SQL commands and comparing resulting storage changes.
- We also present a tool that, given a **disk image** or a **RAM snapshot** can do the following:
  - Identify intact **DBMS pages**, even for multiple DBMSes on the same disk, for all known storage configuration parameters.
  - Recover the **logical schema** (SQL tables and constraints) and all **database table rows** for known parameters (a parameter set will support several different versions of the DBMS, depending on storage changes version-to-version).
  - Extract a variety of **volatile data artifacts** (e.g., deleted rows or pre-update values).
  - Detect evidence of **user actions** such as row insertion order or recently accessed tables.

Paper outline

Fig. 1 shows the high-level architecture overview. In Section "Database storage structure" we review the principles of page-based data storage in relational databases and define the parameters for parsing and recovering these pages. In the same section we also summarize important database-specific storage structures (i.e., non-tables) and discuss the fundamentals of volatile storage and updates. In Section "Deconstructing database storage", we analyze the interesting storage layout parameter trade-offs and explain how these parameters and some user actions can be discovered within a DBMS. Section "Experiments" reports experimental analysis results for a variety of different databases and environment scenarios. Finally, Section "Related work" summarizes related work and Section "Conclusion and future work" contains the conclusions and mentions a number of promising future work directions.

Database storage structure

The storage layer in relational databases partitions all physical structures into uniform pages with a typical size of 4 or 8 KBytes because using a fixed page size significantly simplifies storage and cache management. Page size can be changed by the database administrator, but such a change requires rebuilding data structures: page size cannot be changed for individual tables, at a minimum it is global per tablespace. Two different layers of metadata are involved in database storage: the general information that describes where and how the tables are stored and the per-page metadata for the contents of each individual page. The forensic challenge lies in reconstructing all surviving database content directly from disk (or memory) image using only the metadata included with each page.

From a high level perspective, all relational database pages share the same general structure and break down into three components of interest: the header, the row directory and the row data itself. Depending on the specifics of each database, the page header stores general page information (e.g., *table or an index?* or *which table or index is it?*). This part of the overhead is found at the beginning of the page structure. The row directory component is responsible for keeping track of the row locations as new rows are inserted or old rows are deleted. This row directory may be positioned either between the page header and the row data or at the very end of the page following the row data. The third component is the row data structure that contains the actual page content along with some additional overhead. Fig. 2 shows an overview of how these structures typically interact within a page; the "other structures" area can contain other optional elements only relevant under specific circumstances (e.g., particular kinds of updates). We next describe the comprehensive set of

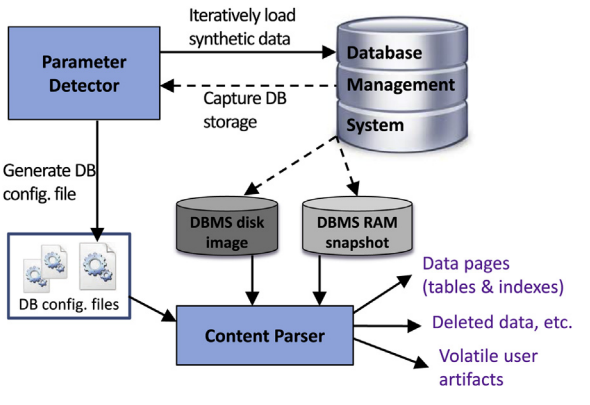

Fig. 1. Overview of parameter detection and data analysis.

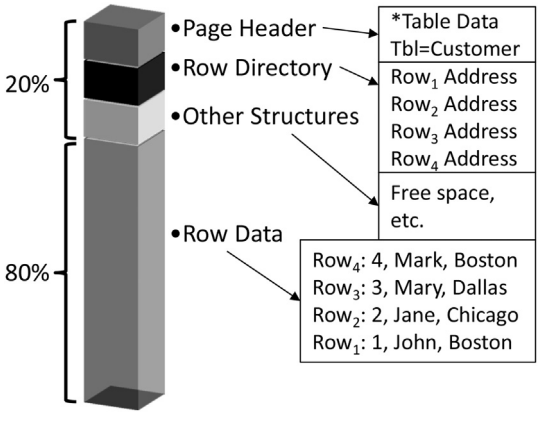

Fig. 2. A structural overview of a database page.