Structural Limitations of B+-Tree forensics

Peter Kieseberg
St. Pölten University of Applied Sciences
St. Pölten, Austria
peter.kieseberg@fhstp.ac.at

Sebastian Schrittwieser
St. Pölten University of Applied Sciences
St. Pölten, Austria
sebastian.schrittwieser@fhstp.ac.at

Edgar Weippl
SBA Research
Vienna, Austria
eweippl@sba-research.org

ABSTRACT

Despite the importance of databases in virtually all data driven applications, database forensics is still not the thriving topic it ought to be. Many database management systems (DBMSs) structure the data in the form of trees, most notably $\mathcal{B}^+$-Trees. Since the tree structure is depending on the characteristics of the INSERT-order, it can be used in order to generate information on later manipulations, as was shown in a previously published approach.
In this work we analyse this approach and investigate, whether it is possible to generalize it to detect DELETE-operations within general INSERT-only trees. We subsequently prove that almost all forms of $\mathcal{B}^+$-Trees can be constructed solely by using INSERT-operations, i.e. that this approach cannot be used to prove the existence of DELETE-operations in the past.

CCS CONCEPTS

• **Security and privacy → Database activity monitoring**;

KEYWORDS

digital forensics, databases, database forensics

ACM Reference Format:
Peter Kieseberg, Sebastian Schrittwieser, and Edgar Weippl. 2018. Structural Limitations of B+-Tree forensics. In ,. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3277570.3277579

1 INTRODUCTION

Digital forensics has become an important factor in the analysis of incidents in the IT world, be it during an official legal investigation, or solely within an internal analysis. While there are many novel approaches in well-researched areas like network or file forensics, the topic of database forensics, i.e. the analysis of databases in order to detect manipulations or steganographically hidden information, has never been in the center of attention, even though it gradually gains importance in the scientific community [10]. Especially considering the proclaimed "age of data science" this seems like a huge blind spot, as most structured data is stored, and often even processed, in some kind of database.
Nevertheless, several approaches for database forensics have been devised in the past. Many of these focus on the extraction of information gathered in log-files or related mechanisms [1], also including NO-SQL databases [9]. Other approaches rely on the analysis of internal mechanisms used for guaranteeing ACID-compliance like the transaction mechanism [8]. Especially for the first strategy, there exists a large body of knowledge targeting many different database management systems like Oracle [13] or MS SQL [17]. One major drawback of most of theses approaches, when compared to those targeting internal mechanisms, is that logs are written for the purpose of detecting malicious behaviour, thus being a primary target for manipulations themselves. Furthermore, users with administrator privileges do have a lot of possibilities regarding log files. This is not so easy with the utilization of internal mechanisms, as manipulations there can possibly destroy the integrity of the database.
Another approach that even works a level of abstraction deeper than the utilization of DBMS-specific internal mechanisms was provided in [12]. In this approach, the authors use the structure of the resulting $\mathcal{B}^+$-Tree that is used to structure the data inside the DBMS in order to detect certain kinds of information. Still, besides the issue of practicability (see [11] for a practical adoption in logging), the approach also requires a certain insertion order of the elements, they have to be entered in a strictly monotonous order with respect to the primary key. In this work we will thus discuss why relaxing this requirement is not easily done and we prove that almost all tree structures as utilized by the original mechanisms can be constructed by solely using INSERT-statements, thus making the detection of DELETEs impossible using this approach.

2 BACKGROUND & RELATED WORK

2.1 $\mathcal{B}$-Trees and $\mathcal{B}^+$-Trees

The $\mathcal{B}$-Tree was originally defined by Bayer [4] as a tree structure where all leaf nodes lie on the same level (balanced tree) and the following properties are obeyed:

- Every node except the root has between $\frac{b}{2}$ and b, the root node between 1 and b elements. The value b is constant and predefined for a given tree ("order" of the tree).
- An inner node with d elements possesses $d + 1$ child nodes.
- The elements inside nodes are sorted.

The difference between the classical $\mathcal{B}$-Tree and the $\mathcal{B}^+$-Tree is that all the payload in a $\mathcal{B}^+$-Tree resides in the leaf nodes, the inner nodes solely hold pointers in order to allow for searching the tree [5].
Insertion in a $\mathcal{B}^+$-Tree works as follows: The leaf node where the elements should be placed is identified. If this node contains less than b elements than the new element is simply added to the internal sorted list of the node. If the leaf already contains b elements, it

CECC 2018, Central European Cybersecurity Conference 2018 (CECC 2018), November 15–16, 2018, Ljubljana, Slovenia
© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-6515-4/18/11…$15.00
https://doi.org/10.1145/3277570.3277579

is split in half with both resulting leafs containing $\frac{b}{2}$ elements (without the new one). This also requires a new key in the parent node, for which the lowest element in the second leaf is selected. This is done iteratively, i.e. in case the parent node now contains more than b elements, it needs to be split too. This can be required to be done until the root node is reached, In case this contains $b + 1$ elements after the insertion, it is split too and a new root node is generated, solely holding the lowest element of the second child node as element.

$\mathcal{B}^+$-Trees are typically used, with slight adaptions, in database storage engines like InnoDB [16]. For example, since full SELECTs are quite common, the leaf nodes are linked with each other in the form of a list, still, the principles of the tree operations stay the same.

2.2 $\mathcal{B}^+$-Tree forensics

While the first work [14] to mention the possibility to infer forensic information from the structure of the $\mathcal{B}^+$-Tree of a database did not give any details on how to retrieve information and what information could be restored, in their work [12], the authors proposed a method for detecting manipulations in databases using the structure of the $\mathcal{B}^+$-Tree of the primary index. The approach has some prerequisite:

- The table is INSERT-only, i.e. there are no UPDATE- and DELETE-operations. While this side-parameter seems quite limiting in nature, it does make sense for a lot of applications in the data warehousing world, especially considering tables for Audit & Control.
- Furthermore, the insertion order of the elements into the table is done strictly monotonous, i.e. the smallest element with respect to the search key is inserted first, then the second-smallest and so on (there are no equally big search keys). This is also quite typical for tables, where the primary key is structured along a timestamp (again very typical for Audit & Control tables)

Given these prerequisites, the structure of the leafs of the resulting $\mathcal{B}^+$-Tree, without considering reorganisations and strictly adhering to the standard insertion routines, has the following form.

Let B be a $\mathcal{B}^+$-Tree with $n > b$ elements which are added in ascending order. Then it holds true that the partition of the resulting k leafs of B has the following structure:

$$n = \sum_{i=1}^{k} a_i, \text{ with } a_i = \frac{b}{2} + 1, \forall i \neq k \text{ and } a_k \geq \frac{b}{2}.$$

In case the structure of the leafs of the $\mathcal{B}^+$-Tree is not adhering to the given formula, manipulations like later INSERTs and changes of older entries can be detected (see Figure 1 for an example).

The forensics approach was also adapted in order to serve as an enhancement for manipulation secure logging like discussed in [11]. The Ficklebase-approach [3] on the other hand discusses mitigation strategies against $\mathcal{B}^+$-Tree-forensics in order to guarantee history independence. In other works [6, 7], the authors discuss the importance of considering internal resources like transaction logs and data structures in databases, also referring to them as potential information leaks and considering $\mathcal{B}^+$-Tree-forensics. $\mathcal{B}^+$-Tree-forensics has also been mentioned as means for detecting evidence related to money laundering and violation of the BCE's KYC policies, as well as evidence extractors [2]. In other work [15], the authors introduce $\mathcal{B}^+$-Tree-forensics into a forensic workflow that is explicitly targeting databases.

$\mathcal{B}^+$-Tree-forensics has also been considered in approaches targeting the modelling of the impact of security policies on compliance, especially regarding the attacker model that needs to be considered [18, 19].

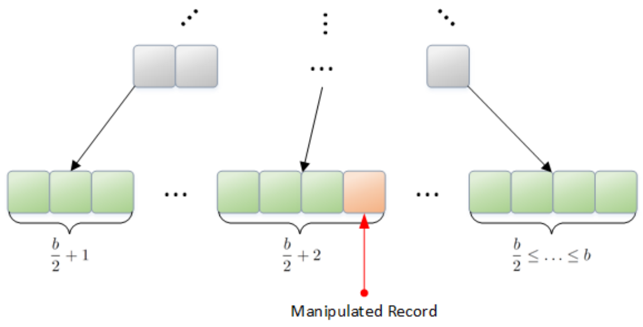

Figure 1: Leafs of a $\mathcal{B}^+$-Tree resulting from ascending ordered inserts only

3 ADOPTION FOR ARBITRARY INSERT-ONLY TABLES

The approach outlined in Section 2, while working in principle, has a major drawback: The insertion strategy for the data is severely limited to (strictly) monotonous INSERT orders. Still, while this might be enough for Audit& Control databases, most data owners want to have more flexibility, thus it would be interesting to extend the approach towards more general INSERT-orders. Thus, we arrive at the research question, whether this approach is able to detect the deletion of elements in an INSERT-only database, where the order of the INSERT-operations is arbitrary.

In order to decide the applicability of this approach, we will discuss, what forms of finite $\mathcal{B}^+$-Trees (i.e. the number of elements in the tree is finite) can be constructed by INSERTS only and where DELETES are needed. For the sake of simplicity, we identify each element with its number when sorted in ascending order, i.e. 4 denotes the 4th smallest element of the set. Additionally, we identify each leaf node with its position in the tree, i.e. a_i is the length of the ith leaf node.

As already outlined in Section 2.2, when doing inserts of n keys sorted in ascending order, we get the following resulting leaf forms:

$$n = \sum_{i=1}^{k} a_i, \text{ with } a_i = \frac{b}{2} + 1, \forall i \neq k \text{ and } a_k \geq \frac{b}{2}.$$

With k being the number of leafs and a_i the number of elements in the i-th leaf.

Our first observation is that we can construct a $\mathcal{B}^+$-Tree where the j-th leaf contains $\frac{b}{2}+2$ elements.

$$n=\sum_{i=1}^{k} a_i,\ \text{with } a_j=\frac{b}{2}+2 \text{ and } a_i=\frac{b}{2}+1, \forall i\neq j,k \text{ and } a_k\geq\frac{b}{2}.$$

Proof. For constructing this tree, we just have to add the elements (which are sorted by size) in the following order:

$$1,\ldots,(j\cdot(\lfloor\frac{b}{2}\rfloor+1)-1),(j\cdot(\lfloor\frac{b}{2}\rfloor+1)+1),\ldots,n,j\cdot(\lfloor\frac{b}{2}\rfloor+1)$$

The first $n-1$ INSERTS will construct a tree like the above structure, where the j-th leaf node has got $\lfloor\frac{b}{2}\rfloor+1$ elements. If we now add element $j\cdot(\lfloor\frac{b}{2}\rfloor+1)$, it won't be inserted at the rightmost node, but in the j-th node (see Figure 2). □

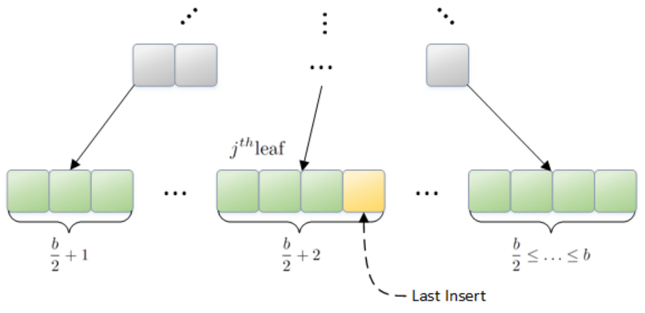

Figure 2: $\mathcal{B}^+$-Tree with one node with $\frac{b}{2}+2$ elements

The above approach can be easily expanded to insert up to b elements in one node by leaving out the suitable elements.

The next question is, if this can be done with an arbitrary number of leaf-nodes.

$$n=\sum_{i=1}^{k} a_i,\ \text{with } a_i\geq\frac{b}{2}+1, \forall i\neq k \text{ and } a_k\geq\frac{b}{2}.$$

Actually, this question is equivalent to the question of building any composition of the number n with particles between $\lfloor\frac{b}{2}\rfloor+1$ and b. For solving this, we adapt the algorithm presented in the last proof with a slight difference (see also Figure 3): The adding of the elements that were left out does not happen in the very end for all nodes at the same time, but in order to guarantee that there are no side-effects, we add these elements soon after the leaf node right to the node they belong to was started.

This can be extended a bit further, in order to construct trees that have leafs with only $\frac{b}{2}$ instead of $\frac{b}{2}+1$ elements in most leaf nodes (except the first and the last leaf), i.e. we can build all leaf-structures of the form

$$n=\sum_{i=1}^{k} a_i,\ \text{with } a_1\geq\frac{b}{2}+1, a_k\geq\frac{b}{2}, a_i=\frac{b}{2}, \forall i\neq 1,k.$$

For proofing this theorem, we just have to proof it is possible for us to generate a leaf holding $\frac{b}{2}$ elements only at each leaf-position

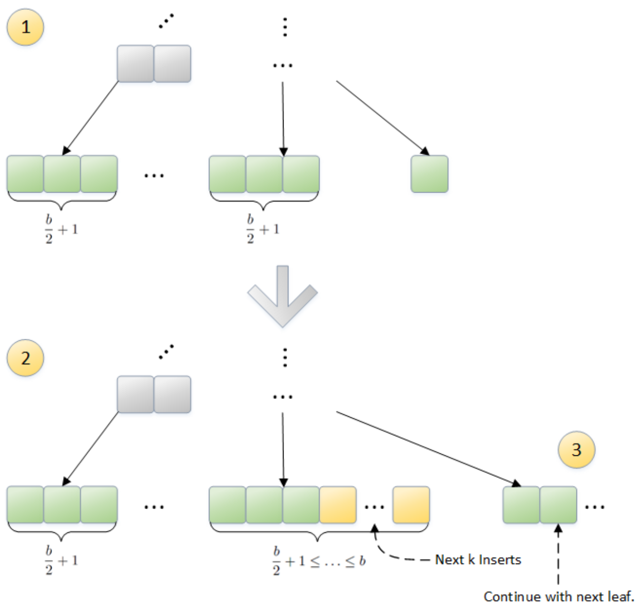

Figure 3: Constructing a $\mathcal{B}^+$-Tree with nodes holding more than $\frac{b}{2}+1$ elements

except the first and the last (Note: The last node always contains the remaining items, its size actually depends on the total number of elements).

Proof. If we have a leaf node i with $a_i=\frac{b}{2}+1$ elements, it is possible to insert a node k with $a_k\geq\frac{b}{2}$ elements left to it by adding these a_k to node i. Since i originally possesses $\frac{b}{2}+1$ elements this sums up to $\frac{b}{2}+1+\frac{b}{2}=b+1$ elements, thus forcing a split of the node. The first $\frac{b}{2}+1=a_i$ elements will stay in node i, thus changing nothing there. The other $\frac{b}{2}$ elements will form the new leaf node (see Figure 4). Thus for constructing a leaf node with only $\frac{b}{2}$ elements, we just need to leaf $\frac{b}{2}$ elements out in the insertion order and add them right after the next leaf node is started. Actually this is not possible for the first node. The proof is rather trivial: According to our prerequisites, in the case of splitting, there are always $\frac{b}{2}+1$ elements left on the right node. So the only possibility of having less elements in the rightmost node at leaf-level is having a tree only consisting of the root node (or doing DELETES).

Additional INSERTS for generating leaf nodes with more than $\frac{b}{2}+1$ elements have to be done after all nodes with $\frac{b}{2}$ elements have been generated, or this will thwart the generation process of the nodes containing only $\frac{b}{2}$ elements. □

The major result of this proof is that there is only one leaf structure of a $\mathcal{B}^+$-Tree where a DELETE-operation is required for construction: When the first leaf node only contains $\frac{b}{2}$ elements.

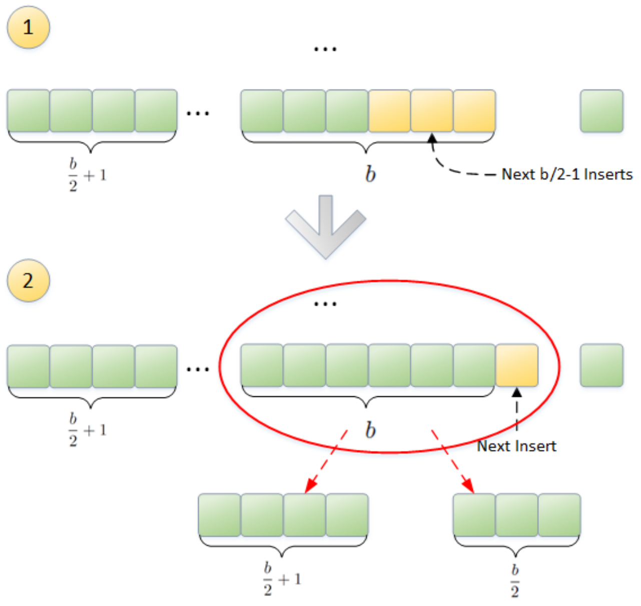

Figure 4: Constructing a $\mathcal{B}^+$-Tree with nodes holding only $\frac{b}{2}$ elements

4 CONCLUSION

In this work we showed that approach for $\mathcal{B}^+$-Tree-forensics for databases as defined in [12] cannot be generalized for the detection of data deletion in case of a table that allows general, non-monotonous, data insertion. Of course, the approach can still be used for the task it was originally intended, indicating manipulations in Audit & Control tables. It must be noted though that, similar to the original approach, we solely concentrated on the structure of the leaf nodes and did not consider the whole structure of the tree including internal nodes. Still, in essence the techniques work similar for changing the inner node structure, just the amount of elements required to be inserted later gets rather large. Furthermore, these proofs can be trivially expanded from $\mathcal{B}^+$-Trees to $\mathcal{B}^*$-Trees, as the only difference between the two is the minimal number of elements inside the leaf nodes, the other requirements remain the same, which also means that the proofs given in this paper work similar. Still, since $\mathcal{B}^*$-Trees are, to the best of our knowledge, not that relevant in database forensics, and were also not part of the original approach, we skipped the details on them.

ACKNOWLEDGEMENTS

This research was funded by the KIRAS program of the Austrian Research Promotion Agency (FFG) and the Josef Ressel Center TARGET. The competence center SBA Research (SBA-K1) is funded within the framework of COMET Competence Centers for Excellent Technologies by BMVIT, BMDW, and the federal state of Vienna. The financial support by the Austrian Federal Ministry for Digital, Business and Enterprise and the National Foundation for Research, Technology and Development is gratefully acknowledged.

REFERENCES

[1] Oluwasola Mary Adedayo and Martin S Olivier. 2015. Ideal log setting for database forensics reconstruction. *Digital Investigation* 12 (2015), 27–40.
[2] Flores Armas and Denys Alberto. 2012. *Guidelines for Collecting Forensic Computing Evidence in order to reinforce the Detection of Money Laundering Activities in the Central Bank of Ecuador.* Master's thesis. University of Derby/2012.
[3] Sumeet Bajaj and Radu Sion. 2013. Ficklebase: Looking into the future to erase the past. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on.* IEEE, 86–97.
[4] R. Bayer and E.M. McCreight. 1972. Organization and maintenance of large ordered indexes. *Acta informatica* 1, 3 (1972), 173–189.
[5] Douglas Comer. 1979. Ubiquitous B-tree. *ACM Computing Surveys (CSUR)* 11, 2 (1979), 121–137.
[6] Denys A Flores, Olga Angelopoulou, and Richard J Self. 2012. Combining digital forensic practices and database analysis as an anti-money laundering strategy for financial institutions. In *Emerging Intelligent Data and Web Technologies (EIDWT), 2012 Third International Conference on.* IEEE, 218–224.
[7] Denys A Flores, Olga Angelopoulou, and Richard J Self. 2013. An Anti-Money Laundering Methodology: Financial Regulations, Information Security and Digital Forensics Working Together. *J. Internet Serv. Inf. Secur.* 3, 1/2 (2013), 101–114.
[8] Peter Frühwirt, Peter Kieseberg, Sebastian Schrittwieser, Markus Huber, and Edgar Weippl. 2013. InnoDB database forensics: Enhanced reconstruction of data manipulation queries from redo logs. *Information Security Technical Report* 17, 4 (2013), 227–238.
[9] WK Hauger and MS Olivier. 2018. NoSQL databases: forensic attribution implications. *SAIEE Africa Research Journal* 109, 2 (2018), 119–132.
[10] Werner K Hauger and Martin S Olivier. 2015. The state of database forensic research. In *Information Security for South Africa (ISSA), 2015.* IEEE, 1–8.
[11] Peter Kieseberg, Sebastian Schrittwieser, Lorcan Morgan, Martin Mulazzani, Markus Huber, and Edgar Weippl. 2013. Using the structure of b+-trees for enhancing logging mechanisms of databases. *International Journal of Web Information Systems* 9, 1 (2013), 53–68.
[12] Peter Kieseberg, Sebastian Schrittwieser, Martin Mulazzani, Markus Huber, and Edgar Weippl. 2011. Trees cannot lie: Using data structures for forensics purposes. In *Intelligence and Security Informatics Conference (EISIC), 2011 European.* IEEE, 282–285.
[13] David Litchfield. 2007. Oracle forensics part 1: Dissecting the redo logs. *NGSSoftware Insight Security Research (NISR), Next Generation Security Software Ltd, Sutton* (2007).
[14] Gerome Miklau, Brian Neil Levine, and Patrick Stahlberg. 2007. Securing history: Privacy and accountability in database systems.. In *CIDR.* Citeseer, 387–396.
[15] Tanushree Shelare and Varsha Powar. [n. d.]. A Database Forensic Approach to Detect Tamper Using B+-Trees. ([n. d.]).
[16] Patrick Stahlberg, Gerome Miklau, and Brian Neil Levine. 2007. Threats to privacy in the forensic analysis of database systems. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data.* ACM, 91–102.
[17] Erin Toombs. 2015. *Microsoft SQL server forensic analysis.* Ph.D. Dissertation. Utica College.
[18] Winfred Yaokumah, Steven Brown, and Alex Ansah Dawson. 2016. Towards modelling the impact of security policy on compliance. *Journal of Information Technology Research (JITR)* 9, 2 (2016), 1–16.
[19] Winfred Yaokumah and Peace Kumah. 2018. Exploring the Impact of Security Policy on Compliance. In *Global Implications of Emerging Technology Trends.* IGI Global, 256–274.