Improving Tiled Bitmap Algorithm for Detection and Analysis of Tampered Database

G. N. Dhanokar¹, S. P. Patil²

Abstract

Web application mainly consists of the database of book store, hospital management, banking etc. These databases can be used for the practically implementation. To access these databases administrator provides the authority to users, but authorized users took the misuse of that authority for performing illegal activity on database & try to hide illegal activity. It is the critical task to find out such illegal activity called tampering. The attacker can leave the evidence behind that can be collected by certain ways by forensic tools for the purpose of further investigations. Tiled bitmap forensic analysis algorithm is used to determine who, when, and what data had been tampered. This algorithm finds out all possible locations of tampered data(s). This paper proposed an approach which finds exact locations of tampered data(s).

Keywords

Security, Algorithms, EDNS

1. Introduction

There are mainly two types of approaches Normal process & Validation. In Normal processing transactions are run and hash values are digitally notarized, and in validation, hash values are recomputed and compared with that previous notarized. If just-computed hash value doesn’t match those previously notarized value at that time tampering is detected. Figure 1 illustrates these two phases. Initially database is running fine, processing many transactions per second. It sends a hash value to the digital notarization service, receiving back a notarization ID that it inserts into the hash sequence. At some time validator will perform validation. The validator, reports that database has been tampered. The DBA and forensic analysis is initiated. The validator provides a vital piece of information, that tampering has taken place, but doesn’t offer much else. Since the hash value is the accumulation of every transaction ever applied to the database, validator can’t understand when the tampering occurred, or what portion of the audit log was corrupted. Actually, the validator does provide a very vague sense of when: sometime before now, and where: somewhere in the data stored before now. Further analysis took place by the forensic analysis algorithm which determines who, when, and what data had been tampered.

Figure 1: Normal Process and Validation
2. System Architecture

System architecture along with the flow of information during normal processing and tamper detection are illustrated in Figure 2 [6]. A user application performs transactions on the monitored database, each of which insert, delete, and update rows of the current state. Behind the scenes, DBMS (an extension of DBMS with transaction-time support) maintains the audit log by rendering a specified relation as a transaction time table. On each modification of a tuple, the DBMS is responsible for hashing the tuples. (The flow of information described is shown with pink arrows.) When a transaction commits, the DBMS obtains a timestamp and computes a cryptographically strong one-way hash function of the tuple data and the timestamp. The hash values obtained from the different transactions are cumulatively hashed and thus linked with each other in order to create a hash chain which at each time instant represents all the data in the database. This chain is termed the total hash chain.

A module called a notarizer sends that hash value to an external digital notarization service (EDNS), which notarizes the hash and returns a notary ID. The notary ID along with the initially computed hash values is stored in a separate smaller MySQL-managed database. (The flow of information described is shown with red arrows.) This database, termed the secure master database, is assumed to exist in a secure site which is in a different physical location from the monitored database.

Figure 2 also shows how tamper detection is achieved. At a later point in time an application called the validator initiates a scan of the entire database and hashes the scanned data along with the timestamp of each tuple. The validator retrieves the previously stored (during notarization) notary ID from the secure master database and sends the information to the EDNS (information flow shown with blue arrows). The EDNS then locates the notarized document/hash using the provided notary ID and checks if the old and the new hash values are consistent. If not, then the monitored database has been compromised. The validator stores the validation result in the secure master database (information flow shown with green arrows). The computation of the total chain, together with the periodic notarizations and validations comprise the normal processing execution phase of the system. Result generated by validator provides a vital piece of information, that tampering has taken place or not.

3. Tiled Bitmap Algorithm

The Tiled Bitmap algorithm [1] uses a logarithmic number of chains for each “tile” of duration $I_N$. The spatial resolution in this case can thus be arbitrarily shrunk with the addition of a logarithmic number of chains in the group. More specifically, the number of chains which constitute a tile is $1 + \log(I_N / R_s)$. It is denoted by the ratio $I_N / R_s$ by $N$, the notarization factor. Value of $N$ is required to be a power of 2. This implies that for all the algorithms, $I_N = N \cdot R_s$ and $R_t = V \cdot I_N = V \cdot N \cdot R_s$. Also, because of the fact that $R_s$ can vary so define $D$ to be the number of $R_s$ units in the time interval from the start until $t_{FVF}$, that is, $D = t_{FVF} / R_s$.

Tiled Bitmap Algorithm may handle multiple CEs but it potentially overestimates the degree of corruption by returning the candidate set with granules which may or may not have suffered corruption(s) (false positive).
Figure 3: Corruption diagram for the TBA

Figure 4: The Bitmap of a Single Tile

4. Proposed Work

In our research work we removed the drawback of previous tiled bitmap algorithm. Tiled Bitmap algorithm was able to find out the possible combination of candidate set which contains false positives. So it was unclear to get exact information about tampered data. In our research we find out the exact information about the tampered data. We developed logic which finds exactly when the tampering took place, where the location of tampered data is and by whom data was tampered as shown in figure 5. When we trace CE on X-axis it should provide the commit time and when we trace CE on Y-axis it should provide exact clock time. During notarization event we are going to calculate the hash value of each transaction that took place between the
notarization intervals \( I_N \) using MD5 algorithm. Then we are storing these calculated values to notarizer table. When we perform detection and analysis at that time we are going to calculate the current hash value of each tuple and then comparing that hash value with the stored hash value in notarizer table. If match did not found then that data is tampered.

```java
// input:
// NH Notarizer Database Hash Values
// CH Current Hash Values
// Dset is the set of Data Field Index
// UN is Username
// DT is date and time
// output:
// Rset the set of Result

Procedure forensic Analysis (NH, CH, Dset, UN, TD, Rset)
1: Initially Result Set Empty Rset=
2: for i = 1 to total number of data fields
3:   CHi Current Hash Value of Di
4:   NH Notarized Hash Value of Di
5:   if CHi != NH
6:      Rset = Rset + Di
7: end of for
8: Return Rset
```

**Figure 6: The Proposed Algorithm**

Our proposed algorithm is as shown in figure 6. To find out the result we perform following procedure. During detection and analysis we calculate the current hash value (CHi) of each tuple and then comparing that hash value with the stored hash value in notarizer table (NHi). If match did not found then that data is tampered. The data field index is then stored to Rset. In this way the complete tuples are checked to find the correct tampered fields. This algorithm finally returns the set of all the tampered data (Rset). Details of the tampering is discovered using log which actually set the user name as he/ she login into the system and by using date and time operation on the same case calculate the accurately when the tampering took place.

Figure 7 present a function for computing possible values about tampered data. We defined different functions as follows PossibleValues, getRightMostGenerateFunction, FunkySort. The PossibleValues function is used to find out possible corrupted locations of tampered data. On line 2 (figure7), the getRightMost helper function is called to preprocess the target binary number and to fill the rightmost array in order to answer the “rightmost zero” query in constant time. On line 5 (figure 7), the GenerateFunction is called recursively which creates the candidate set elements. On line 6 (Figure 7), we call the sorting function. This funkySort function creates the sequence of indices which will result in the ordering of the candidate set elements.

```java
// Ckt is an array of Binary Numbers
// p is the position of one of the zeros in target bit number
1: Function PossibleValues(String Ckt, int p)
2:   RightMostArray =getRightMost(Ckt);
3:   for(int i=0;i<RightMostArray.size();i++)
4:     int p1=(Integer) RightMostArray.get(i);
5:     Generate (Ckt,p1);
6:   FunkySort(FinalSet);
```

**Figure 7: The Possible Values Function**

5. Result

Table 1 shows the running time for forensic analysis algorithms. We assume that the spatial detection resolution Rs is equal to 1 for simplicity. Observe that the algorithms become progressively slower because of the increased number of chains maintained and used during forensic analysis. The Monochromatic Algorithm, while being the fastest algorithm, suffers from the fact that only the first corruption event can be detected. As noted, the Tiled Bitmap Algorithm can be slightly optimized by retaining the cumulative chain of the Monochromatic in order to locate the first corrupted tile by performing binary search, although this refinement does not affect its asymptotic running time. When we compare our approach with all the three of above ours is faster as in our model we never going to do the multiple corruptions in post operation of corruption event. In our case we keep finding the corruption at each and every transaction. In this complexity calculation of algorithms D denotes the no of days and Iv is the validation time interval.

The forensic cost is a function of D (expressed as the number of Rs units), N, the notarization factor (with \( I_N = N \cdot Rs \)), V, the validation factor (with \( V = I_v / I_N \)), and k the number of corruption sites (the total number of \( t_i, t_b, t_p \)'s). A corruption site differs from a CE because a single timestamp CE has two corruption sites. FC(D,N, V, k) = NormalProcessing(D,N, V) + ForensicAnalysis(D,N, V, k) + AreaP (D,N, V, k) + AreaU(D,N, V, k)

NormalProcessing, is the number of notarizations and validations made during normal processing in a span of D days. The second component, ForensicAnalysis, is the cost of forensic analysis in terms of the number of validations made by the algorithm to yield a result. Note that this is different from the running time of the algorithm. The rationale behind this quantity is
that each notarization or validation involves an interaction with the digital notarization service, which costs real money.

The third and fourth components informally indicate the manual labor required after automatic forensic analysis to identify exactly where and when the corruption happened. This manual labor is very roughly proportional to the uncertainty of the information returned by the forensic analysis algorithm. It turns out that there are two kinds of uncertainties, formalized as different areas. That these components have different units than the first two components is accommodated by the weights.

In order to make the definition of forensic cost applicable to multiple corruption events it need to distinguish between three regions within the corruption diagram. These different areas are the result of the forensic analysis algorithm identifying the corrupted granules. This distinction is based on the information content of each type.

- Area \( P \) or corruption positive area is the area of the region in which the forensic algorithm has established that corruption has definitively occurred.
- Area \( U \) or corruption unknown area is the area of the region in which we don’t know if or where a corruption has occurred.
- Area \( N \) or corruption negative area is the area of the region in which the forensic algorithm has established that no corruption has occurred.

Each corruption site is associated with these three types of regions of varying area. The stronger the algorithm the less costly it is, with smaller Area \( P \) and Area \( U \). It is also desirable that Area \( N \) is large but since TotalArea is constant this is achieved automatically by minimizing Area \( P \) and Area \( U \). As in our proposed algorithm is identifying exactly where and when the corruption happened as shown in our corruption diagram (Figure 5) so definitely the Area \( N \) will be the region other than corrupted region.

As our Area \( N \) is larger than all other algorithms so the cost of our approach is less. Table 2 shows the cost for each of the forensic algorithms assuming a spatial detection resolution of one hour (Rs=1) and a single corruption event. In this case, we observe the opposite trend compared to the one observed for the running times of the algorithms. For a sufficiently large validation interval \( Iv \), the Tiled Bitmap Algorithm has the smaller cost. This is because the ratio \((1+\log Iv)/ Iv\) becomes less than one. When we compare values of tiled bitmap algorithm with our approach, \((\log Iv)/ Iv\) yields even smaller value than tiled bitmap. So we can state that our approach is having smallest cost of all algorithms. Figure 8 shows the results of the experimental cost validation. The experiments used the following setup: \( D = 1 \) to 256, \( Rs = 1 \), and \( Iv = 8 \) using the cost formulas in order notation (as given in Table 2).

### Table 1: Running Time Complexity of Algorithms

<table>
<thead>
<tr>
<th>S. N.</th>
<th>Algorithm</th>
<th>Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Monochromatic</td>
<td>( O(\log(D/Iv)) )</td>
</tr>
<tr>
<td>2</td>
<td>RGB</td>
<td>( O(D/Iv) )</td>
</tr>
<tr>
<td>3</td>
<td>Tiled Bitmap</td>
<td>( O(D(1+\log Iv)/Iv+D) )</td>
</tr>
<tr>
<td>4</td>
<td>Proposed Approach</td>
<td>( O(\log(D/Iv)) )</td>
</tr>
</tbody>
</table>

![Figure 8: The cost of the Algorithms](image)

### Table 2: Worst case cost/space complexity of Algorithms

<table>
<thead>
<tr>
<th>S. N.</th>
<th>Algorithm</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Monochromatic</td>
<td>( O(D) )</td>
</tr>
<tr>
<td>2</td>
<td>RGB</td>
<td>( O(D) )</td>
</tr>
<tr>
<td>3</td>
<td>Tiled Bitmap</td>
<td>( O(D(1+\log Iv)/Iv) )</td>
</tr>
<tr>
<td>4</td>
<td>Proposed Approach</td>
<td>( O(D(\log Iv)/Iv) )</td>
</tr>
</tbody>
</table>

6. Conclusion

Database Forensics is an important topic that has not received much research attention. The approach is based on cryptographically one way hashing function, notarization service, and validator. Tiled Bitmap algorithm was able to find out the possible combination of candidate set which contains false positives. This research finds out the exact information about the tampered data with the help of
There are no commercially available tools for doing effective database forensics. The attacker can leave the evidence behind that can be collected by certain ways by forensic tools for the purpose of further investigations. In future work it is a good opportunity to develop such a commercial tool for doing effective database forensics.

References


Ganesh N. Dhanokar received the BE degree in IT from Amravati University in 2010. He is currently pursuing PG under the guidance of Ms. S. P. Patil from North Maharashtra University.

Sonal Patil received the ME degree in CSE from TIT Bhopal. She is currently working as Assistant Professor, Department of CSE, G. H. Raisoni Institute of Engineering And Management Jalgaon.