International Journal of Advanced Computer Research (ISSN (print): 2249-7277 ISSN (online): 2277-7970)
Volume-2 Number-4 Issue-6 December-2012

Improved Tiled Bitmap Forensic Analysis Algorithm

C. D. Badgujar[1], G. N. Dhanokar[2]
G. H. Raisoni Institute of Engineering and Management Jalgaon

Abstract

In Computer network world, the needs for security and proper systems of control are obvious and find out the intruders who do the modification and modified data. Nowadays Frauds that occurs in companies are not only by outsiders but also by insiders. Insider may perform illegal activity & try to hide illegal activity. Companies would like to be assured that such illegal activity i.e. tampering has not occurred, or if it does, it should be quickly discovered. Mechanisms now exist that detect tampering of a database, through the use of cryptographically-strong hash functions. This paper contains a survey which explores the various beliefs upon database forensics through different methodologies using forensic algorithms and tools for investigations. Forensic analysis algorithms are used to determine who, when, and what data had been tampered. Tiled Bitmap Algorithm introduces the notion of a candidate set (all possible locations of detected tampering(s)) and provides a complete characterization of the candidate set and its cardinality. Improved tiled bitmap algorithm will cover come the drawbacks of existing tiled bitmap algorithm.

Keywords

Temporal databases, EDNS, DBMS,

1. Introduction

This section summarizes the tamper detection approach. There are several related ideas that in concert allow tamper detection.

- The first insight is that the DBMS can maintain the audit log in the background, by rendering a specified relation as a transaction-time table. This instructs the DBMS to retain previous tuples during update and deletion, along with their insertion and deletion/update time (the start and stop timestamps), in a manner completely transparent to the user application. An important property of all data stored in the database is that it is append-only: modifications only add information; no information is ever deleted. Hence, if old information is changed in any way, then tampering has occurred. Oracle supports transaction-time tables with its workspace manager.
- The second insight is that the data modified (inserted/ updated/deleted) by a transaction can be cryptographically hashed to generate a secure one-way hash of the transaction.
- The third insight is to digitally notarize this hash value with an external notarization service. So even if the intruder has full access to the database itself, the DBMS, and even the operating system and hardware, the intruder cannot change the hash value. This makes it exceedingly difficult to make a series of changes to the audit log that generates the same hash value.

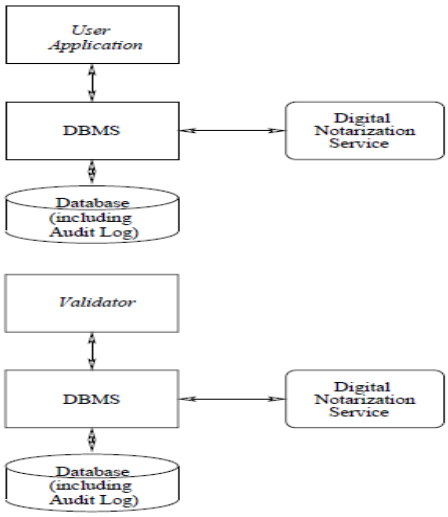

Fig 1: Normal Operation and Audit Log Validation

This basic approach differentiates two execution phases: normal processing, in which transactions are run and hash values are digitally notarized, and validation, in which the hash values are recomputed and compared with that previous notarized. It is during validation that tampering is detected, when the just-computed hash value doesn't match those

previously notarized. Figure 1 illustrates these two phases.

Initially database is running fine, processing many transactions per second. Periodically, say every night at midnight, it sends a hash value to the digital notarization service, receiving back a notarization ID that it inserts into the hash sequence. At some validator will perform validation. The validator, reports that our 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.)

2. System Architecture

System architecture along with the flow of information during normal processing and tamper detection are illustrated in Figure 2.
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 periodically sends that hash value, as a digital document, to an external digital notarization service (EDNS) such as Surety (www.surety.com), 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.

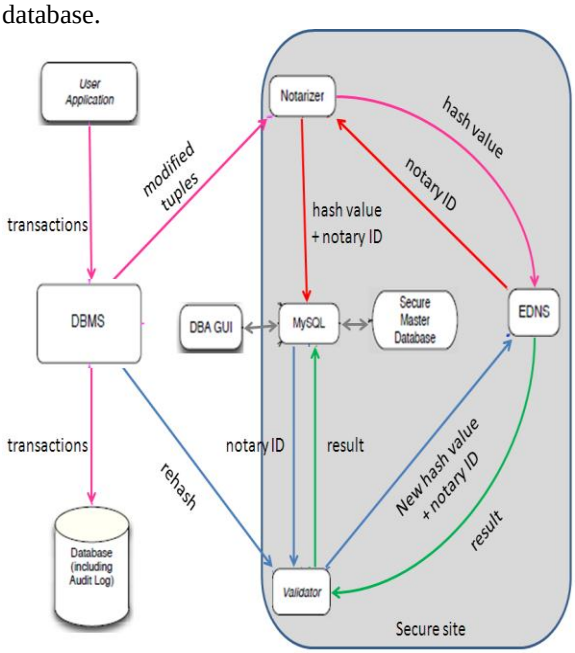

Fig 2: System Architecture for Normal Processing

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.

3. Tiled Bitmap Algorithm

Validation provides a single bit of information: has the database been tampered with? To provide more information about when and what, we must hash the data of various sequences of transactions during validation. The database transactions are hashed in commit order creating a *hash chain*. Then, during forensic analysis of a subsequent validation that

detected tampering, those chains can be rehashed to provide a sequence of truth values (success or failure), which can be used to narrow down "what."

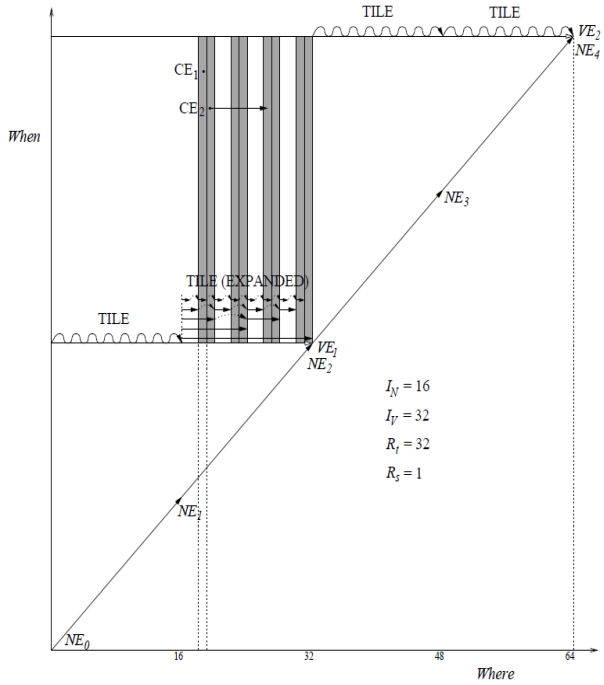

Fig 3: Corruption diagram for the Tiled Bitmap Algorithm

Table 1: Description of symbols

Symbol	Name	Definition
CE	Corruption event	An event that compromises the database
VE	Validation event	The validation of the audit log by the notarization service
NE	Notarization event	The notarization of a document (hash value) by the notarization service
I_V	Validation interval	The time between two successive *VEs*
I_N	Notarization interval	The time between two successive *NEs*
R_t	Temporal detection resolution	Finest granularity chosen to express temporal bounds uncertainty of a CE
R_s	Spatial detection resolution	Finest granularity chosen to express spatial bounds uncertainty of a CE
t_{FVF}	Time of first validation failure	Time instant at which the CE is first detected
USB	Upper spatial bound	Upper bound of the spatial uncertainty of the corruption region
LSB	Lower spatial bound	Lower bound of the spatial uncertainty of the corruption region
UTB	Upper temporal bound	Upper bound of the temporal uncertainty of the corruption region
LTB	Lower temporal bound	Lower bound of the temporal uncertainty of the corruption region

The Tiled Bitmap algorithm 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 $1 + \lg(I_N / Rs)$. It is denoted by the ratio I_N / Rs by N, the notarization factor. Value of N is required to be a power of 2. This implies that for all the algorithms, $I_N = N \cdot Rs$ and $Rt = V \cdot I_N = V \cdot N \cdot Rs$. Also, because of the fact that Rs can vary so define D to be the number of Rs units in time interval from the start until t_{FVF}, $D = t_{FVF} / Rs$. 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 (false positives). Figure 3 shows that the Tiled Bitmap Algorithm will produce a candidate set with the following granules: 19, 20, 23, 24, 27, 28, 31, 32. The corruptions occur on granules 19, 20 and 27 while the rest are false positives.

4. Example

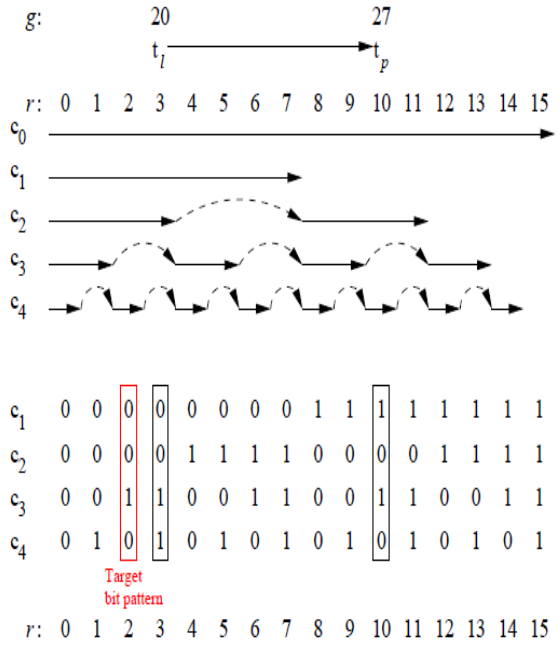

Fig 4: The Bitmap of a Single Tile

Let us turn to an example involving a corruption. Consider CE1 in Figure 3. When the first tile is found in which a corruption has occurred via binary search in order to locate t_{RVS}. In this figure CE1 has $t_l = 19$ and a relative position within the second I_N of 2. If

validator validate the hash chains of the tile in which the CE transpired then validator get the string 00010 (most significant bit corresponds to the chain which covers all the units in I_N), termed the target bit pattern. The numerical value of the target string 00010 is 2 which is exactly the relative position of the granule within the second I_N.

Now, let's see what happens if a timestamp corruption occurs and both t_l and *tp* are within the same tile. Figure 3 also shows a postdating CE2 with t_l = 20 and *tp* = 27 which are both in the second tile (I_N = 16). If each of these were to appear on their own the target bit patterns produced by the tile validation would be 0011 (3rd granule within N) and 1010 (10th granule within N). However, since both occur at the same time within the same I_N and the hash chains are linked together, then the bit patterns given above are ANDed and the target 0010 is the actual result of the validation, as shown in Figure 4. This target corresponds to the existence of the two suspect days t_l and *tp* without being able to distinguish between the two.

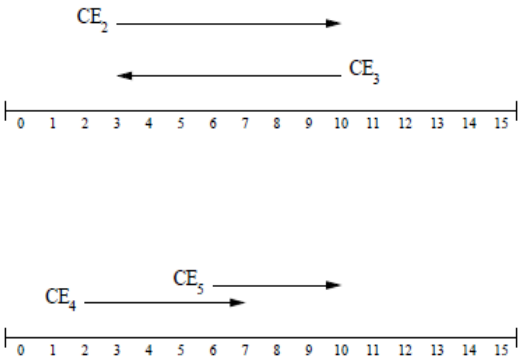

Fig 5: CEs resulting in the same target bit pattern

In reality the situation is more involved: when dealing with multiple CEs there might be many combinations of bit patterns which when ANDed can yield the target bit pattern computed during forensic analysis. Thus even in the simple case where a single post/backdating CE does not have its endpoints in different tiles can introduce ambiguity. For example, analyzer can't distinguish between the two scenarios shown in Figure 5 because in both cases the target bit pattern is the same. In the first case, both CE2 and CE3 produce the target bit pattern 0010 because the AND operation is commutative: 0011 ∧ 1010 = 1010 ∧ 0011. For this reason analyzer cannot distinguish between CE2 and CE3. Moreover, distinguishing between CE2, CE3 and CE4, CE5 is also impossible because CE4 and CE5 also produce the same target bit pattern as before. More specifically, CE4 produces the bit pattern 0010 ∧ 0111 = 0010 and CE5 produces again 0110 ∧ 1010 = 0010.

5. Proposed Work

Existing Tiled bitmap algorithm can simply find out the possible combination of candidate set which contains false positives. So it is unclear to get exact information about tampering. Our improved tiled bitmap algorithm will be able to find out the exact information about the tampering of data as shown in figure 6. When we project CE on X-axis it should provide the commit time and when we project CE on Y-axis it should provide exact clock time.

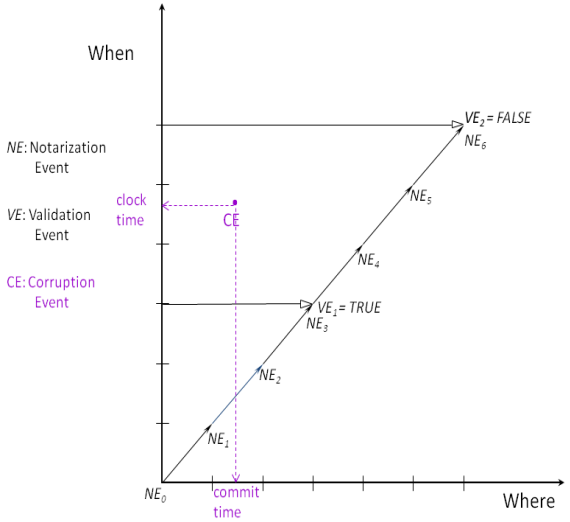

Fig 6: Improved Tiled Bitmap Algorithm

The partial hash chains within a tile are denoted by $C_0(T)$, $C_1(T)$,…. $C_{lg(Iv)}(T)$; with $C_i(T)$ denoting the i^{th} hash chain of the tile which starts at time instant T. On line 4, the algorithm iterates through the different tiles and checks if the longest partial chain $C_0(T)$ evaluates to FALSE. If not, it moves on to the next tile. If the chain evaluates to FALSE (line 5), the algorithm iterates through the rest of the partial chains in the tile (line 7) and "concatenates" the result of each validation to form the target number (line 8). Then, the candidate Set function is called (line 9) to compute the entire candidate set elements from the target number. On lines 10–12, the candidate granules are renumbered to reflect their global position. The function renumber( ) on line 11 uses Rs to find the global position of r, computing g as a

single granule, or group of granules if Rs > 1. Once the C_{set} is reported, the administrator can exactly pinpoint the corrupted tuples.

```
// input:  τ_FVF is the time of first validation failure
//         I_V is the validation interval
//         k is used for the creation of C_{t,k}
//         R_s is the spatial detection resolution
// output: C_set, an array of binary numbers
function Tiled_Bitmap(τ_FVF, I_V, k, R_s)
1:  t ← 0        // the target
2:  C_set ← C_temp ← ∅
3:  τ ← 1
4:  while τ < τ_FVF do
5:    if ¬ val_check(c_0(τ)) then
6:      n ← lg(I_V/R_s)
7:      for i ← n to 1
8:        t ← t + 2^{n−i}·val_check(c_i(τ))
9:      C_temp ← candidateSet(t, n, k)
10:     for each r ∈ C_temp
11:       g ← renumber(r, τ, R_s)
12:       C_set ← C_set ∪ {g}
13:   τ ← τ + I_V
14: return C_set
```

Fig 7: Tiled Bitmap Algorithm

6. Conclusion

We have seen that the existence of multi-locus CEs can be better handled by summarizing the sites of corruption via candidate sets, instead of trying to find their precise nature. We proceed now to develop a new algorithm that avoids the limitations of all the previous algorithms and at the same time handles the existence of multi-locus CEs successfully.

References

[1] Kyriacos E. Pavlou and Richard T. Snodgrass, "The Tiled Bitmap Forensic Analysis Algorithm," IEEE transaction on knowledge and data engineering, Vol. 22, pp no.590-601, April 2010.

[2] Nina Godbole and Sunit Belapure, "Cyber Security, Understanding Computer Forensics and Legal Perspectives", Wiley-India, 2011.

[3] Harmeet Kaur Khanuja and D.S.Adane, "Database Security Threats and Challenges in Database Forensic: A Survey," International Conference on Advancements in Information Technology With workshop of ICBMG 2011.

[4] Jayshree T. Agale & Shefali .P.Sonavane, "Hash Based Intrusion Detection and Forensic Analysis of Tampered Database," ICCSIT-10th June, 2012.

[5] Kyriacos E. Pavlou and Richard T. Snodgrass, "Forensic Analysis of Database Tampering," ACM Transactions on Database Systems, September 2008.

[6] By Aaron C. Newman, CTO & Founder, "Security Auditing In Microsoft SQL Server", Application Security, Inc, 2005.

[7] Richard T. Snodgrass, Shilong Stanley Yao and Christian Collberg, "Tamper Detection in Audit Logs," Proceedings of the 30th VLDB Conference, Toronto, Canada, 2004.

Chandrashekhar Badgujar received the ME degree in CSE from NMIMS, Shirpur. He is currently working as Assistant Professor, Department of CSE, G. H. Raisoni Institute of Engineering And Management Jalgaon.

Ganesh Dhanokar received the BE degree in IT from Amravati University in 2010. He is currently pursuing PG under the guidance of Mr. C. D. Badgujar from North Maharashtra University.