IMPROVING DAMAGE ASSESSMENT EFFICACY IN CASE OF FREQUENT ATTACKS ON DATABASES

Prahalad Ragothaman and Brajendra Panda
Department of Computer Science and Computer Enginering, University of Arkansas, Arkansas, USA

Abstract: *A database log is the primary resource for damage assessment and recovery after an electronic attack. The log is a sequential file stored in the secondary storage and it can grow to humongous proportions in course of time. Disk I/O speed dictates how fast damage assessment and recovery can be done. To make the process of damage assessment and recovery more efficient, segmenting the log based on different criteria has been proposed before. But the trade off is that, either segmenting the log involves a lot of computation or damage assessment is a complicated process. In this research we propose to strike a balance. We propose a hybrid log segmentation method that will reduce the time taken to perform damage assessment while still segmenting the log fast enough so that no intricate computation is necessary. While performing damage assessment, we re-segment the log based on transaction dependency. Thus during repeated damage assessment procedures, we create new segments with dependent transactions in them so that the process of damage assessment becomes faster when there are repeated attacks on the system.*

Key words: Database log, damage assessment, log segmentation, transaction dependency

1. Introduction

Ever since the dawn of the Internet, there have been reports of unauthorized entry into computer systems and rendering them inconsistent and unstable. There are many methods to protect a system from such attacks but savvy hackers always find newer ways to break into a system and use it maliciously. Several intrusion detection mechanisms have also been developed. But those methods do not detect an intrusion as soon as it occurs. This results in the damage caused by the malicious user to spread throughout the system in an exponential manner. In course of time, the system may become so unstable that we have to shut the entire system down in order to bring it back to a consistent state. This is highly unacceptable in time critical database systems where valid users must have access to data at every, and all times. Hence the next best solution to the problem would be to design fast and efficient damage assessment and recovery algorithms to be used during the post intrusion detection scenario.

Traditional logging mechanisms as described in [2], [5], and [3], record only the "write" operations of a transaction. A traditional log does not suffice to recover a database from an attack, as it does not contain the "read" operations of the transaction. Read operations are essential to establish dependences among transactions and data items.

To expedite the process of damage assessment and recovery after an attack, the log can be segmented based on certain criteria. That way, when an attack is detected, we can skip large portions of the log, which, we are sure, does not contain malicious or affected transactions.

Several log segmentation approaches have been devised. Notable among them are the transaction dependency based approach [8], the data dependency based approach [10], and segmenting based on certain criteria like fixed number of transactions, time window for transactions to commit and space for committed transactions [11]. In this research, we propose to devise a hybrid method of log segmentation that uses the approaches presented in [8] and [11] so that damage assessment and hence recovery can be expedited.

The rest of the paper is organized as follows. In section 2, the prior work and the motivation behind this research are described. In section 3, description of log re-segmentation with accompanying cases and algorithms are presented. In section 4, the damage assessment model using the re-segmented log is described. Section 5 presents the simulation results. Section 6 concludes the paper.

2. Prior Work and Motivation

In [4], the researchers have proposed several guidelines for trusted recovery. Amman et al. [1] followed a transaction dependency approach that uses relationships among transactions to identify and repair damage in the database. Panda and Giordano [9] adopted a data dependency approach to recover from malicious attacks. Reordering transactions for efficient recovery has been discussed by Liu et al. [6]. A distributed recovery approach has been offered by Liu and Hao in [7]. But all of these approaches scan a sequential log file, which is very huge. Log segmentation techniques using transaction dependency and data dependency were presented in [8] and [10] respectively. These methods segment the log file in such a way that all dependent transactions (or dependent data-items in the case of data dependency segmentation) are stored in one segment. By doing so, it can be made sure that we do not have to scan a large portion of the log when an attack is detected.

But a major drawback in these approaches is that they use valuable system resources to perform intricate computation to determine dependences among transactions or data-items while the execution of the transactions is still on. Also, there is a chance of a segment growing too large because too many transactions or data-items may be dependent on one another. Different segments have a chance of merging into one segment and ultimately it could be one huge segment, as big as the log itself. This defeats the purpose of log segmentation. In [11] researchers have presented methods of segmenting the log based on number of transactions, a time window for transactions to commit and fixed space for committed transactions. In the first method, a segment is formed after a fixed number of transactions commit. In the second approach, there is a time window provided for transactions to commit. All transactions that committed in that time frame form one segment. In the third method, the space for a segment remains fixed. All committed transactions that fit into that segment are stored. Transactions are not allowed to span from one segment to another. Thus the size of each segment is kept under control and running the risk of a segment growing too big is avoided in all these three approaches. Also, each of these methods uses very little computation while segmenting the log. However, a significant amount of computation has to be done to determine the dependences among the segments during damage assessment. In scenarios where attacks are quite frequent, this method may not yield the fastest solution to recover a database.

In this research, we present a hybrid method of log segmentation that uses the techniques provided in [11] and [8]. We propose to further segment a log already segmented based on any of the three approaches described in [11] based on transaction dependency approach as described in [8]. We shall do the re-segmentation while assessing damage during subsequent attacks on the database. By doing so, we intend to achieve a significant improvement in terms of time required during damage assessment while still keeping the log segmentation algorithm simple enough so that the time for execution of transactions is not hindered.

3. Hybrid Log Segmentation

Our model is based on the following assumptions: (a) Transaction operations are scheduled in accordance with the rigorous two-phase locking protocol as defined in [3], (b) Read operations are also recorded in the log file, (c) Intrusion is detected using one of the intrusion detection techniques and the *id* of the attacking transaction is available, (d) The log is never purged, and (e) Blind writes are not allowed. Below, a list of definitions is presented that are helpful in understanding the research and the algorithms.

Definition 1: Transaction T_j is said to be dependent on transaction T_i if T_j read one or more data items that was previously written by the committed transaction T_i.

Definition 2: A *tuft* is a group of transactions that adheres to any one of the three models presented in [11]. The transactions in a *tuft* are stored in the chronological order in which they committed. It is represented as Γ_i where 'i' denotes the *tuft* number.

Definition 3: A *read_items* list is a list of all the data items that were read by all the transactions in a segment.

Definition 4: A w*rite_items* list is a list of all the data items that were written by all the transactions in a segment.

Definition 5: A *tuft_table* is a table that contains the transaction number and the tuft in which it is present.

Definition 6: An *affected_items* list contains all the data items that were written either by a malicious or an affected transaction.

Definition 7: A transaction is said to be affected if it updates the value of a data item using the value of another data item that was previously written by either a malicious or another affected transaction.

Definition 8: A "size-controlled-segment" is a segment that was created using one of the three approaches described in [11]. In other words, we choose to call a *tuft* as a size-controlled-segment.

Definition 9: A "size-un-controlled-segment" is a segment that was created using transaction dependency.

The log segmented based on any of the three approaches described in [11] can be pictorially represented as shown in Figure 1.

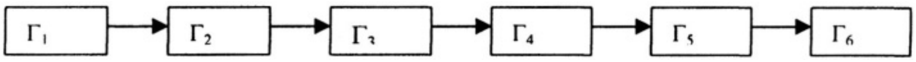

Figure 1: Log Segments before Re-segmentation

Re-segmenting the log begins after an attack is detected and the attacking transaction is available. The process is done during the damage assessment phase. Let us assume that an attack was detected in Γ_2. It has been shown that scanning of transactions in Γ_1 is unnecessary because none of the transactions in Γ_1 read a data item written by any of the transactions present in Γ_2. Rigorous two-phase locking protocol ensures this. The attacking transaction in Γ_2 is determined using the *tuft_table*. All transactions in all the segments until the last affected segment, i.e. the segment that has the last affected transaction in it, are scanned. A new segment is started with the first malicious transaction in it. To determine the dependency among transactions, we intersect the "write" set of the malicious transaction with the read set of the transaction to be scanned. There are two cases, as discussed below, depending upon the results of the intersection.

3.1 The result is not a null set

The transaction is affected. That particular transaction is stored in the newly formed segment that contains the malicious transaction and other affected transactions. The data items that were written by the transaction are appended to the *affected_items* list. Thus all malicious and affected transactions will be present in one segment at the end of the damage assessment phase and all affected data items in the *affected_items* list.

3.2 The result is a null set

This means that the transaction did not read any affected data. The "read" and the "write" sets of the transaction are stored in a new *read_items* list and *write_items* list respectively. We can be sure that the transaction is not dependent on any other transaction in any of the other size-un-controlled segments that were formed. Hence a new segment is created and the operations of the transaction are stored in it.

During subsequent scanning of transactions, the "read" items of that transaction is intersected with all the *write_items* lists and *affected_items* list available to determine if there is a dependency between the transaction and the segments. By doing so, it is checked whether the transaction read a data item that was previously written by another transaction. If the result of all the intersections is a null set, it means that transaction is completely independent of all other transactions scanned so far. A new segment is created and the operations of that transaction are stored in it. Also, a new *read_items* list and a *write_items* list is started as in the previous case and the "read" set and the "write" set of the transaction are stored into the respective lists.

If on the other hand, the result of the intersection of the "read" set of the transaction with two or more *write_items* lists is not a null set, it means that the transaction is dependent upon two or more transactions that are present in two different segments. In such a case, those two segments are merged and the operations of the current transaction are stored in the merged segment. The *read_items* list and the *write_items* list are also merged and the "read" set and the "write" set of the transaction are stored into the appropriate lists.

If the transaction is dependent on only one of the segments that have been formed, it is added to the end of the segment upon which it is dependent. The "read" set and the "write" of the transaction are stored into the appropriate lists. Thus one or more size-un-controlled segments will be present in parallel between Γ_1 and the last affected segment. Each size-un-controlled segment will have its own *read_items* list and a *write_items* list. A pictorial representation of a re-segmented log is shown in Figure 2.

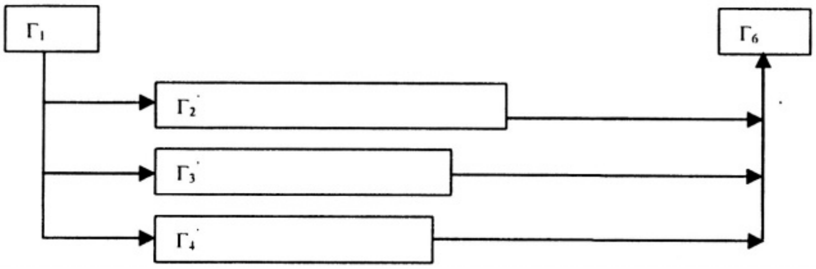

Figure 2: Log Segments after Re-Segmentation

Some segments may be larger or smaller than others depending on how many transactions are present in that segment. Each of the newly formed segments Γ_2', Γ_3', and Γ_4' will contain transactions that are dependent on one another. When an attack is detected, only the segment containing the malicious transaction has to be scanned since all affected transactions would be present in that segment alone. All other size-un-controlled segments can be safely avoided. We present an algorithm for log re-segmentation in the following section.

3.3 Algorithm for log re-segmentation during damage assessment

1. Determine the position of the attacking transaction using the *tuft_table.* Let us assume that the transaction, say T_i, is present in Γ_i.
2. Set *affected_items* = write_set(T_i); *read_items* = read_set(T_i).
3. Start new segment Γ_i'.
4. For each transaction, say T_j, that appears after the attacking transaction T_i, in Γ_i until the last transaction in the last affected *tuft,* say Γ_j, where $j > i$

   If (*affected_items* $\cap$ read_set(T_j) != ϕ)

```
        Add $T_j$ to $\Gamma_i$'; Add the write set of $T_j$ to affected_items; Add read_set( $T_j$ ) to read_items.
    Else
        Intersect the write set of $T_j$ with all available read_items list. If none are available, start a new size-un-controlled-segment, add the operations of $T_j$ in the segment, start a new read_items list and a write_items list and add the read_set and write_set of $T_j$ into the respective lists. Continue from step 3.
            If the result of all the intersections is a null set
                Start a new size-un-controlled-segment, say $\Gamma_j$', and add the operations of $T_j$.
                Start a new read_items list and a write_items list and add the read set and the write set of $T_j$ into the respective lists.
            If the result of the intersection is not a null set with only one of the segments
                Add the operations of $T_j$ into that segment.
                Update the appropriate read_items list and the write_items list.
            If the result of the intersection is not a null set with more than one of the segments
                Merge the segments into one single segment.
                Add the operations of $T_j$ into the merged segment.
                Merge the appropriate read_items list and write_items list and add $T_j$'s read and write items into the respective lists.
```

As it is evident in the above method, there is the risk of having to manage a segment that is too large because various segments might get merged to form one big segment. Eventually this segment might end up being as big as the log itself. In order to avoid this scenario, a new method to segment the log in a hybrid manner is proposed. In this approach, pointers are provided to link the information flow from one segment to another instead of merging the segments together. Thus after subsequent damage assessment on the database, the log can pictorially be represented as shown in Figure 3.

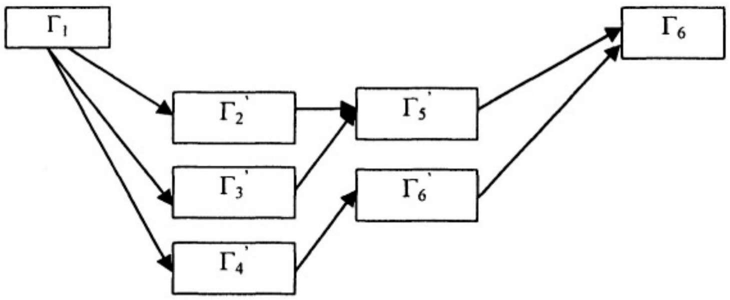

Figure 3: A Newer Method to Segment the Log in a Hybrid Manner

Let us consider the segments as shown in Figure 1. Assume that an attack was detected in Γ_2. As mentioned before, none of the transactions that are present in Γ_1 need to be scanned as they are not affected. Damage assessment is carried out as mentioned before and all the cases hold true here too. Let us assume that the first set of parallel segments are formed and they are named as Γ_2', Γ_3' and Γ_4'. During subsequent scanning of transactions from other size-controlled segments, it is assumed that a transaction read data items that were written by transactions from two different size-un-controlled segments making that transaction dependent on two different segments. From Figure 3, it is evident that a transaction present in Γ_5' read data items written by transactions in Γ_2' and Γ_3'. Thus pointers are established from these two segments to Γ_5' to show that there is an information flow from both Γ_2' and Γ_3' onto Γ_5'. Thus during subsequent damage assessment procedures, the pointers can be checked and the information flow can be obtained. An algorithm to segment the log using the new hybrid log segmentation approach is given below.

3.4 Algorithm for log re-segmentation using the new method of hybrid log segmentation

```
1. Determine the tuft, say $\Gamma_i$, where attacking transaction, say $T_i$, is present.
2. Set *affected_items* = write_set($T_i$); *read_items* = read_set($T_i$); Start new
   segment $\Gamma_i'$.
3. For each transaction, say $T_j$, that appears after the attacking transaction $T_i$,
   in $\Gamma_i$ until the last transaction in the last affected *tuft*, say $\Gamma_j$, where j > i
      If ( *affected_items* $\cap$ read_set($T_j$) != $\phi$ )
         Add $T_j$ to $\Gamma_i'$; Add the write set of $T_j$ to *affected_items*; Add
         read_set( $T_j$ ) to *read_items*.
      Else
         Intersect the write set of $T_j$ with all available *read_items* list.
         If none are available, start a new size-un-controlled-segment, add
         the operations of $T_j$ in the segment, start a new *read_items* list and
         a *write_items* list and add the read_set and write_set of $T_j$ into the
         respective lists. Continue from step 3.
           If the result of all the intersections is a null set
              Start a new size-un-controlled-segment, say $\Gamma_j'$, and add the
              operations of $T_j$.
              Start a new *read_items* list and a *write_items* list and add the
              read set and the write set of $T_j$ into the respective lists.
           If the result of the intersection is not a null set with only one of
           the segments
              Add the operations of $T_j$ into that segment.
```

```
        Update the appropriate *read_items* list and the *write_items* list.
    If the result of the intersection is not a null set with more than one
    of the segments
        Establish pointers between the two segments.
        Retain the *read_items* list and *write_items* list as it is.
```

4. Damage Assessment Using the Re-Segmented Log File

There are two cases to consider during damage assessment process when an attack is detected after re-segmentation. They are explained based on the segments shown in Figure 3.

4.1 An attack is detected in Γ_1 or in any of the size-controlled segments that were ignored when damage assessment was done the first time

The operations of all the transactions from the point of attack in Γ_1 until the point where the size-un-controlled segments start are scanned. The "write" set of the first malicious transaction in Γ_1 is added to the *affected_items* list. The *affected_items* list is then intersected with the "read" set of transactions that appear after the malicious transaction in all the *tufts* until the size-un-controlled-segments start. The log gets re-segmented with all dependent transactions in one segment. Cases similar to those described in the previous section hold good here too. Thus new sets of parallel size-un-controlled segments are formed. Dependency between each of the newly formed size-un-controlled-segments and the existing size-un-controlled segments is then established. It has to be noted that the segments will not be merged as described in the previous section. Instead, pointers will be established to determine information flow. An algorithm to assess damage for the case discussed is given below.

4.1.1 Algorithm for damage assessment for the case described above

```
1. Let the first attacking transaction in $\Gamma_1$ be $T_i$. Add the write_set of $T_i$ to
   the *affected_items* list. Start a new size-un-controlled-segment, say $\Gamma_1$'
   and add the operations of $T_i$ in $\Gamma_1$'.
2. For every transaction that appears after $T_i$, say $T_j$, until the last transaction
   before the size-un-controlled-segment starts, do
      If ( *affected_items* $\cap$ read_set( $T_j$ ) != $\phi$ )
         Add write_set( $T_j$ ) to *affected_items*; Add the operations of $T_j$ to $\Gamma_1$'.
      Else
```

```
Intersect the read_set of $T_j$ with all available *read_items* lists that
were formed for the newly created size-un-controlled-segments.
If none are available, start a new size-un-controlled-segment and add
the operations of $T_j$ to that segment. Start new *read_items* list and
*write_items* list and update them accordingly.
    If the result of all the intersections is a null set
        Start a new size-un-controlled-segment and add the operations
        of $T_j$.
        Start a new *read_items* list and a *write_items* list and add the
        read set and the write set of $T_j$ into the respective lists.
    If the result of the intersection is not a null set with only one of
    the segments
        Add the operations of $T_j$ into that segment.
        Update the appropriate *read_items* list and the *write_items* list.
    If the result of the intersection is not a null set with more than one
    of the segments
        Establish appropriate pointers between segments.
        Record the "read" items and the "write" items in the
        appropriate lists.
```

4.2 An attack is detected in any of the size-un-controlled segments

In this case, all the size-controlled segments that appear prior to the segment in consideration and all other size-un-controlled-segments that were formed with the current segment can be safely ignored, as there would be definitely no dependency between those segments. Thus, the damage assessment process begins by scanning each transaction after the first malicious transaction in the current segment. This is followed checking every size-controlled-segment and size-un-controlled-segment that appears after the first scanned segment until the last affected segment in the log file. With the help of pointers from one segment to another, we can determine the information flow and thus know what segments need to be scanned after the current one. If there are pointers from a segment leading to two different segments, both the segments have to be scanned after the current segment is scanned. Similarly, if two different pointers from two different segments lead to one single segment, then that segment must be scanned twice while assessing damage. If the result is not a null set, it means that segment is affected and one or more transactions in that segment have read a data item that was previously written by a malicious or affected transaction. An algorithm for this case is presented below.

4.2.1 Algorithm for damage assessment for the case described above

```
1. Identify the size-un-controlled-segment where the malicious transaction
   is present. Let us assume it is $\Gamma_i$'. All other size-un-controlled-segments
   parallel to $\Gamma_i$' can be ignored. Identify the malicious transaction, say $T_i$,
   in $\Gamma_i$.
2. Append the write_set of $T_i$ and all other transactions in the same
   segment to the affected_items list.
3. For each segment, say $\Gamma_j$, that appears after the current segment until the
   last affected segment, do
      If ( affected_items $\cap$ read_items( $\Gamma_j$ ) != $\phi$ )
         If $\Gamma_j$ is a size-un-controlled segment
            Merge $\Gamma_i$' and $\Gamma_j$.
            Merge the respective read_items list and write_items list.
            Append the write_items list to the affected_items list.
         If $\Gamma_j$ is a size-controlled-segment then for each transaction, say $T_k$
         in $\Gamma_j$
            If ( affected_items $\cap$ read_items( $T_k$ )!= $\phi$ )
               Append operations of $T_k$ to $\Gamma_i$'.
               Append the read_set and write_set of $T_k$ to appropriate
               read_items list and write_items list.
            Else
               Intersect the read_set of $T_k$ with all available write_items
               lists of size-un-controlled-segments that were recently
               created.
               If the result of the all the intersections is a null set or if no
               size-un-controlled-segments were recently formed
                  Create a new size-un-controlled segment with the
                  operations of $T_k$ in it.
                  Start a new read_items list and a write_items list and
                  add the read_set and write_set of $T_k$ to the appropriate
                  list.
               If the result of the intersection is not a null set with only one
               segment
                  Append the operations of $T_k$ and its read_set and
                  write_set to the segment, the read_items list and
                  write_items list respectively.
               If the result of the intersection is not a null set with more
               than one segment
                  Establish pointers between the appropriate segments.
```

In the case when an attack is detected in any of the size-controlled-segments that appears after all the size-un-controlled segments, the scenario is similar to that described in section 3.

5. Simulation and Results

A “C“ program was developed to simulate a database log file that conforms to the rigorous two-phase locking protocol. The log was segmented using the fixed number of transactions approach with 50 transactions in each tuft. The algorithms developed in this research were then implemented into a program and applied on the segmented log file. Table 1 shows some of the parameters used to draw the chart depicted in Figure 4.

Table 1: Values of Parameters for Chart Shown in Figure 4

Total number of transactions	500
Total number of data items	5000
Maximum data items accessed by a transaction	30

The values obtained under the “traditional approach” legend were determined by assuming an attacking transaction having transaction *id* 150, and then determining how many bytes of data are read from the log while performing damage assessment without having a segmented log. The “number” legend shows the number of bytes read from a log segmented based on the number of transactions and assuming an attacking transaction with transaction *id* 150 too. The third legend, which is “Hybrid 1”, shows the number of bytes read from a log which was re-segmented based on the hybrid approach presented in this research. An attacker was assumed and damage assessment was done once and a log segmented based on the hybrid approach was obtained. With this log, and the same attacking transaction, the algorithm was implemented again but the log was not segmented as it is proposed in this research. The number of bytes read was determined. This process was carried out several times by changing the seed of the program thus obtaining a new log file. An average of all the runs was calculated and the chart was obtained.

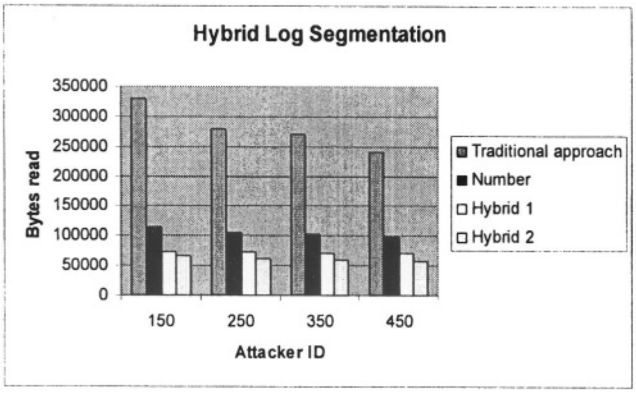

Figure 4: Comparison of Various Damage Assessment Procedures with the Hybrid Approach Using the Values Shown in Table 1

"Hybrid 2" shows the number of bytes read by the damage assessment program when it was implemented as proposed in this research. An attacking transaction with transaction *id* 50, was assumed to be stored in a log that is segmented based on the number of transactions. Damage assessment was performed and a log segmented based on the hybrid approach was obtained. With this log as reference, four different attacking transactions with ids 150, 250, 350 and 450 were assumed. Each time an attack was assumed, damage assessment was done and while doing so, the log was re-segmented based on transaction dependency using the pointer method that was discussed earlier. An average of all the runs was taken. Subsequently, attackers 150, 250, 350 and 450 were assumed and each time the same procedure was carried out. A different log was obtained each time by changing the seed in the program. Figure 5, shows a graph obtained using parameters given in Table 2.

Table 2: Values of Parameters for Chart Shown in Figure 5

Total number of transactions	500
Total number of data items	5000
Maximum data items accessed by a transaction	40

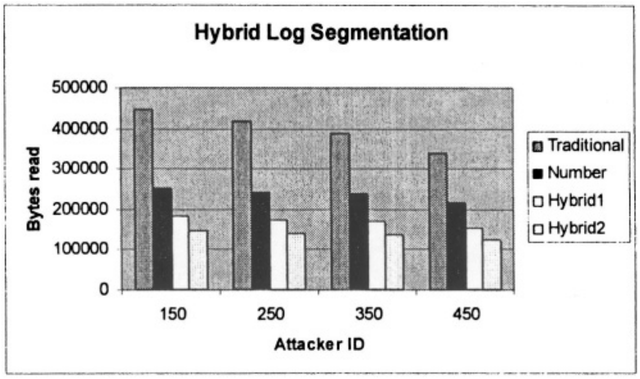

Figure 5: Comparison of Various Damage Assessment Procedures with the Hybrid Approach Using the Values Shown in Table 2

6. Conclusion

In this research, we have presented methods of re-segmenting an already segmented log based on transaction dependency for much faster damage assessment and hence recovery. We have overcome the shortcomings of the previous work where we observed that damage assessment would be more time consuming when compared to other log segmentation approaches like transaction dependency and data dependency while still being much faster had there been no segmentation at all. The model that we have presented here will work best in scenarios where attacks are more frequent. The segmented log will be re-segmented again based on transaction dependency thus limiting damage to only one segment. The process of re-segmenting is done while performing damage assessment and thus system resources will not be wasted. Different cases were observed while re-segmenting and assessing damage using the re-segmented log. Each case was discussed in detail and algorithms were provided to handle the cases separately. The algorithms were implemented in a simulation model and the results were discussed. It was also shown that our model performs better during damage assessment than when the log is not segmented at all or when the log is segmented using number of transactions. The model is also expected to perform similarly well on a log segmented based on either time or space.

Acknowledgement

This work has been supported in part by US AFOSR under grant F49620-01-10346. The authors are thankful to Dr. Robert. L. Herklotz for his support, which made this work possible.

References

[1] P. Amman, S. Jajodia, C. D. McCollum, and B. Blaustein, Surviving Information Warfare Attacks on Databases, *Proceedings of the 1997 IEEE Symposium on Security and Privacy,* May 1997.

[2] P.A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems,* Addison-Wesley, 1987.

[3] R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems, Third Edition,* Addison-Wesley, 2000.

[4] S. Jajodia, C. D. McCollum, and P. Amman, Trusted Recovery, *Communications of the ACM,* 42(7), pp. 71-75, July 1999.

[5] H. F. Korth, A. Silberschatz, and S. Sudarshan, *Database System Concepts, Third Edition,* McGraw-Hill International Edition, 1997

[6] P. Liu, P. Ammann, and S. Jajodia, Rewriting Histories: Recovering from Malicious Transactions, *Distributed and Parallel Databases,* 8(1), pp. 7-40, January 2000.

[7] P. Liu and X. Hao, Efficient Damage Assessment and Repair in Resilient Distributed Database Systems, *Proceedings of the 15th Annual IFIP WG 11.3 Conference on Database and Application Security,* July 2001.

[8] B. Panda and S. Patnaik, A Recovery Model for Defensive Information Warfare, *Proceedings of the 9th International Conference on Management of Data,* p. 359-368, Hyderabad, India, December 1998.

[9] B. Panda and J. Giordano, Reconstructing the Database After Electronic Attacks, *Database Security XII: Status and Prospects,* S. Jajodia (editor), Kluwer Academic Publishers, 1999.

[10] B. Panda and S. Tripathy, Data Dependency Logging for Defensive Information Warfare, *Proceedings of the 2000 ACM Symposium on Applied Computing,* p. 361 – 365, Como, Italy, March 2000.

[11] P. Ragothaman, and B. Panda, Modeling and Analyzing Transaction Logging Protocols for Effective Damage Assessment, *Research Directions in Data and Applications Security,* E. Gudes and S. Shenoi (editors), Kluwer Academic Publishers, 2003.