

# Prevention of Malicious Transactions in Database Management Systems

A thesis submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology

*in*

Computer Science and Engineering

*by*

Himanshu Gahlaut



Department of Computer Science and Engineering

National Institute of Technology Rourkela

May 2009

# Prevention of Malicious Transactions in Database Management Systems

A thesis submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology

*in*

Computer Science and Engineering

*by*

Himanshu Gahlaut

*under the guidance of*

Prof. Sanjay Kumar Jena



Department of Computer Science and Engineering

National Institute of Technology Rourkela

May 2009



## National Institute of Technology Rourkela

### CERTIFICATE

This is to certify that the thesis entitled, “PREVENTION OF MALICIOUS TRANSACTIONS IN DATABASE MANAGEMENT SYSTEMS” submitted by HIMANSHU GAHLAUT in partial fulfillment of the requirements for the award of Bachelor of Technology Degree in COMPUTER SCIENCE AND ENGINEERING at the National Institute of Technology, Rourkela is an authentic work carried out by him under my supervision and guidance.

And to the best of my knowledge, the matter embodied in the thesis has not been submitted to any other University/Institute for the award of any Degree or Diploma.

PLACE: NIT Rourkela

**Dr. Sanjay Kumar Jena**

DATE:

Professor

National Institute of Technology

Rourkela – 769008

## **ACKNOWLEDGEMENT**

I wish to express my deep sense of gratitude and indebtedness to Prof. Sanjay Kumar Jena, Department of Computer Science and Engineering, N.I.T. Rourkela for introducing the present topic and for his inspiring guidance, valuable suggestions and support throughout this project work.

I am thankful to all my Professors and Lecturers and members of the department for their generous help in various ways for the completion of the thesis work.

I would love to thank my family members for encouraging me at every stage of this project work. Last but not least, my sincere thanks to all my friends who have patiently extended all sorts of help for accomplishing this undertaking.

Himanshu Gahlaut

# Table of Contents

1 Introduction .....	1
2 Related Work .....	5
2.1 Existing Intrusion Detection Systems.....	5
3 A new approach for intrusion detection in DBMS .....	8
3.1 Bloom Filters .....	8
3.2 Counting Bloom Filters.....	10
3.3 Prevention Model .....	10
3.4 Intrusion Prevention System.....	12
3.4.1 Profiling the transactions and assigning to users .....	12
3.4.2 Giving weight to commands and setting the CBF's for transactions .....	13
3.4.3 Instant Detection and Prevention.....	13
4 Experimental Evaluation .....	16
4.1 Experimental Setup.....	16
4.2 Experimental Results and discussion .....	17
4.2.1 False Negatives .....	17
4.2.2 Coverage .....	22
5 Conclusion and Future Work .....	24

## List of Figures

Fig. 1 DB system using the DBMTD Mechanism.....	6
Fig. 2 Bloom filter with four Hash Functions.....	9
Fig. 3 Counting Bloom Filter.....	10
Fig. 4 Database Management System with Intrusion Prevention System.....	11
Fig. 5 Typical profiles of database transactions.....	13
Fig. 6 Profile of the TPC-C transactions.....	17
Fig. 7 variation of False Negatives % with increase in no. of hashing functions for different sizes of CBF (s).....	19
Fig. 8 variation of False Negatives % with increase in size of CBF for different odd no. of hashing functions used.....	20
Fig. 9 variation of False Negatives % with increase in size of CBF for different even no. of hashing functions used.....	21
Fig. 10 Repetitive Patterns.....	22
Fig. 11 Coverage.....	23

# ABSTRACT

Database Management Systems are a key component in the information infrastructure of most organizations nowadays so security of DBMS has become more crucial. Several mechanisms needed to protect data, such as authentication, user privileges, encryption, and auditing, have been implemented in commercial DBMS. But still there are some ways through which systems may be affected by malicious transactions. Our definition of malicious transaction is that transaction which the user is not authorized to perform. Even the sequence of the operations in the transaction is not to be violated. Existing intrusion detection systems use logs to detect malicious transactions. Logs are the histories of the transactions committed in the database. The disadvantage of using logs is that they require lot of memory. In addition to this sometimes even after a transaction is detected as malicious it cannot be rolled back. In this thesis we present a method by which we can overcome the uses of logs and can detect malicious transactions before they are committed. We use counting bloom filters to store the sequence of commands in a transaction and use a prevention model for instant detection of malicious transactions. Simulation was carried out for a single user providing sequence of queries varying the size of the CBF from 1 to 15 and no. of hashing functions from 1 to 15. It was concluded that by choosing optimal value of size of CBF and number of hashing functions the detector can be made to prevent a malicious transaction with a probability of almost 99.85%.

# **Chapter 1**

## **Introduction**

# Chapter 1

## 1 Introduction

Database security is the system, processes, and procedures that protect a database from unintended activity. Unintended activity can be categorized as authenticated misuse, malicious attacks or inadvertent mistakes made by authorized individuals or processes. *Database security* is also a specialty within the broader discipline of computer security.

Information is the most critical resource for many organizations [1]. In many cases, the success of an organization depends on the availability of key information and, therefore, on the systems used to store and manage the data supporting that information. The protection of data against unauthorized access or corruption due to malicious actions is one of the main problems faced by system administrators. Due to the growth of networked data, security attacks have become a dominant problem in practically all information infrastructures.

Security breaches can be classified as unauthorized data observation, incorrect data modification and data unavailability [2]. Unauthorized data observation results in the disclosure of information to users not intended to gain access to such information. Incorrect modification of data either intentional or unintentional, results in an incorrect database state. Data unavailability results in improper and untimely functioning of the organization.

A DBMS allows users to define the data to be stored in terms of a data model, which is a collection of high-level metadata that hide many low-level storage details. Most DBMS are based on the relational data model [3]. The relational data model defines a database as a collection of relations, where each relation is a table with rows and columns. The consistency of the data stored in different tables is assured by a set of relational integrity mechanisms, including referential integrity mechanisms that assure data in one table is consistent with data in other tables. A typical database application is a client-server system where a number of users are connected to a DBMS via a terminal or a desktop computer (the trend today is to access database

servers through the internet using a browser). The user actions are translated into SQL commands by the client application and sent to the database server. The results are sent back to the client to be displayed in the adequate format by the client application. In many cases, the system includes an application server that runs business rules code (i.e., code directly related to data application handling) and performs load balancing of the multiple client sessions. The main goal of security in DBMS is to protect the system and the data from intrusion, even when the potential intruder gets access to the machine where the DBMS is running. To protect the database from intrusion the DBA must prevent and remove potential attacks and vulnerabilities. The system vulnerabilities are an internal factor related to the set of security mechanisms available (or not available at all) in the system, the correct configuration of those mechanisms (which is a responsibility of the DBA), and the hidden flaws on the system implementation. Vulnerability prevention consists on guarantying that the software used has the minimum vulnerabilities possible. This can be achieved by using adequate DBMS software. On the other hand, as the effectiveness of the security mechanisms depend on their correct configuration and use, the DBA must correctly configure the security mechanisms by following administration best practices. Vulnerability removal consists on reducing the vulnerabilities found in the system. The DBA must pay attention to the new security patches release by software vendors and install those patches as soon as possible. Furthermore, any configuration problems detected on the security mechanisms must be immediately corrected.

Security is an concept that includes the following properties [1]: authenticity (guarantees that a service or piece of information is authentic), confidentiality (absence of unauthorized disclosure of a service or piece of information), integrity (protection of a service or piece of information against illicit and/or undetected modification), and availability (protection of a service or piece of information against possible denials of service caused by malicious actions). In database management system the above solutions are met with different components or methods. Initially to gain access to a database a user has to provide identification and authentication. Identification is the means by which a user provides a claimed identity to the system (i.e. using a username). Authentication is the means of establishing the validity of this claim (password). Issuing identification and initial authentication key to the users is the job of DBA. The user can change his identification and authentication to enter into the database later on.

Access Control Methods [4] ensures confidentiality in DBMS. If anybody tries to access the data object the Access Control Methods checks for the rights of the user against the data object with the set of authorizations stated by the Administrators. Access is the ability to do something with a computer resource. This usually refers to a technical ability (e.g. read, modify, delete, execute or use an external connection). Authorization is the permission to the computer resource.

To ensure Data Integrity the access control mechanism and semantic integrity constraints are used together [2]. Whenever a subject tries to modify some data, the access control mechanism verifies that the user has the right to modify the data, and the semantic integrity subsystem verifies that the updated data are semantically correct. Semantic correctness is verified by a set of conditions, or predicates, which must be verified against the database state. The Recovery subsystem and the Concurrency control mechanism ensures that data is available and correct despite hardware and software failures and accesses from concurrent application programs.

The main goal of database security mechanisms is to protect the data stored in the database from unauthorized accesses or malicious actions in general. In spite of all these steps taken to secure the database still there are possibilities for hackers to take the chance of breaking into the system. A hacker may get a password in many ways and try to operate the system. Unfortunately, the risk of malicious actions and attacks in DBMS becomes a real threat in such a situation and the proposal of instant intrusion detection mechanisms for DBMS is a logical and relevant step. That is just the goal of the present work.

Typical database security attacks can be classified as: 1) intentional unauthorized attempts to access or destroy private data; 2) malicious actions executed by authorized users to cause loss or corruption of critical data (e.g., system manager that have legitimate access to database servers but should not access database data); and 3) external interferences aimed to cause undue delays in accessing or using data, or even denial of service. In the present work we are particularly interested in the first two types of attacks, which in practice correspond to malicious transactions executed by authorized users or by unauthorized users that gain access to the database by exploring system vulnerabilities.

As database security mechanisms are not design to primarily detect intrusions (are designed to avoid the intruder's access to database data), there are many cases where the

execution of malicious sequences of SQL commands (transactions) cannot be detected (or avoided).

The following points present some examples:

- The DBA does not activate the necessary security mechanisms (e.g., authentication, user privileges, data encryption, auditing, etc), which allows intruders to get access to database data.

- The security mechanisms available are incorrectly configured permitting potential intruders (*hackers*) to access the database.

- Hidden flaws in the database implementation may allow hackers to connect to the database server by exploring those defects.

- Unauthorized users “still” the credentials of authorized users in order to access the database server.

- Authorized users take advantage of their privileges to maliciously access or destroy data.

Malicious transactions may damage the database integrity and availability. However, in spite of the pertinence of the detection of malicious database transactions, the reality is that very few practical mechanisms which are able to identify users executing malicious transactions has been proposed so far.

This thesis proposes a new mechanism for the detection of malicious transactions in DBMS. As in a typical database environment it is possible to define the profile (sequence of SQL commands) of all the transactions that each user is allowed to execute, the proposed mechanism uses that profile of valid transactions to identify user’s attempts to execute invalid sequences of commands. The sequences of commands that constitute each valid transaction are obtained by analysis of the execution profile of the database client applications.

# **Chapter 2**

## **Related Work**

**Existing Intrusion Detection Systems**

# Chapter 2

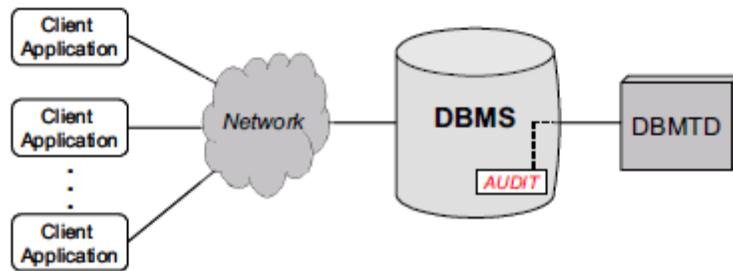
## 2 Related Work

### 2.1 Existing Intrusion Detection Systems

In 2005 Marco and Henrique [1] proposed a Database Malicious Transaction Detector (DBMTD). DBMTD mechanism is a log based mechanism for the detection of malicious transactions in DBMS. In practice malicious database transactions are related to security attacks carried out either externally or internally to the organization. **External security attacks** are intentional unauthorized attempts to access or destroy the organization's private data. These attacks are perpetrated by unauthorized users (*hackers*) that try to gain access to the database by exploring the system vulnerabilities (e.g., incorrect configuration, hidden flaws on the implementation, etc). On the other hand, **internal security attacks** are intentional malicious actions executed by authorized users. These users use their normal privileges to execute illicit actions for their personal benefit or to harm the organization by causing loss or corruption of critical data. Given the growing threat represented by security attacks to databases, and the fact that databases are where most of the relevant data of organizations is actually stored, a practical mechanism to detect the execution of malicious transactions in databases is of utmost importance. However, to the best of our knowledge, none of the existing commercial DBMS provides such a mechanism. Manual supervision and audit procedures are normally the only tools the DBA can use to detect potential database intrusion. In a typical database environment the profile of the transactions that each user is allowed to execute is usually known by the DBA, as the database transactions are programmed in the database application code. In other words, the transactions are not ad hoc sequences of SQL commands. On the contrary, database transactions are well defined sequences of commands performing a finite set of predefined actions. For example, in a banking application users can only perform the operations available at the

application interface (e.g., withdraw money, check account balance, etc). No other operation is available for the end users. Particularly, end users cannot execute ad hoc SQL commands.

The DBMTD mechanism uses the profile of the transactions defined in the database applications (authorized transactions) to identify user attempts to execute malicious transactions. DBMTD is built on top of the auditing mechanism implemented by most commercial DBMS. The audit log is used by DBMTD to obtain the sequence of commands executed by each user, which is then compared with the profile of the authorized transactions to identify potential malicious transactions.



**Figure 1. DB system using the DBMTD mechanism.**

IDS are based basically on two models [2] Anomaly Model and Misuse Model. Anomaly model establishes a normal activity profile for the system and if any activity fails to match the profile of the normal profile then the IDS considers it as an intrusion attempt. The misuse model is based on the assumption that there are ways to represent attacks in the form of a pattern or a signature so that even variation of the same attack can be detected. Here the IDS maintain a database of all the known attack signatures. It raises an alarm whenever the attack signature matches the one that the IDS have in its database. There are possibilities that the IDS might be unable to detect an intrusion attempt (false negative) or might catch a normal behavior as intrusion (false positive). IDS are of three types namely Network based, Host based, combined IDS. A network based IDS deployed outside the firewall monitors the data packets traveling over the network and any possible attack on the data packets to modify or read them are recorded by the IDS. A host based IDS is deployed on the host machine.

One more Intrusion detection system was proposed in [5] which explained how to identify malicious transaction by checking for data dependency. Before any updating of data, it

has to be read and after updating it has to be written back. There exists the pre-write set and post write set for a updating. Then identification of a malicious transaction is done by comparing the consistency in the pre-write set and post-write set of the user transactions. The detection model proposed in [1, 5] used to detect the malicious transaction after the transaction has been committed.

Another intrusion detection model was proposed in [2] which detect the malicious transactions before they are committed. It considers a situation when an intruder performs a malicious transaction by transferring some amount from account A to account B without the intervention of the account A's owner. Before the intrusion is detected and repaired the money may be drawn away. This transaction cannot be repaired. It is better to prevent a malicious transaction than detecting a curing it later which is an additional burden. Logs are used in recovery purposes to maintain the ACID properties. Logs are difficult to manage especially when systems of different systems are using. Logs cannot be employed in embedded systems. Since logs are usually stored in buffer if buffer is turned off then the logs are lost. If at all the logs are stored in the secondary storage the time is taken more to fetch and compare with the transaction profile. If a masquerader gets a log, he can know all the sequence of work done by the user. Considering above drawbacks an Intrusion Prevention System without using logs was proposed in [2]. It's a space efficient method of implementation for detecting a malicious transaction since it do not use logs for the detector.

We follow the model proposed in [2] in our present work for intrusion detection. In [2] sequence in counting bloom filters were used for intrusion detection. In our present work we prove by our experimental evaluation that the same prevention model can work just by using simple counting bloom filters and still it can be made to detect malicious transactions 99.85% times if optimal size of CBF and optimal no. of hashing functions are used.

# **Chapter 3**

## **A new approach for intrusion detection in DBMS**

**Bloom filter**

**Counting Bloom Filter**

**Prevention Model**

**Intrusion Prevention System**

# Chapter 3

## 3 A new approach for intrusion detection in DBMS

### 3.1 Bloom Filters

A Bloom filter is a method for representing a set of elements (also called keys) to support membership queries. It was invented by Burton H. Bloom in 1970 [6].

An empty Bloom filter is a bit array of  $m$  bits, all set to 0. There must also be  $k$  different hash functions defined, each of which maps or hashes some set element to one of the  $m$  array positions. In order to add an element, the element is supplied to each of the  $k$  hash functions to get  $k$  array positions and the bits at all these positions are set to 1. A simple Bloom Filter with 4 hash functions is shown in Fig. 2 [6].

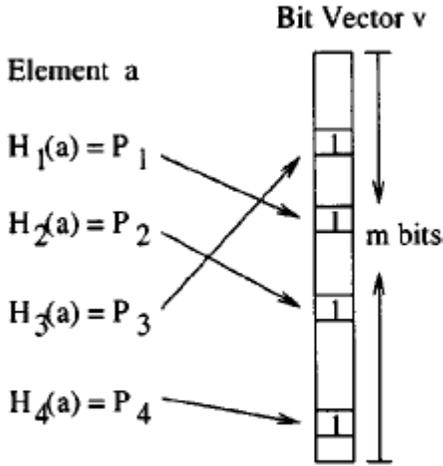


Fig.2 Bloom Filter with four hash functions.

Example: Choose  $m = 6$  (number of bits) and  $k = 2$  (no of hash functions).

Let the hash function taken be  $h1(x) = x \text{ mod } 6$

$$h2(x) = (2x + 3) \text{ mod } 6$$

We first initialize the Bloom filter B [1::6], and then insert 9 and 11 in a set A.

	h1 (x)	h2(x)	Bloom Filter (B)
Initialize:			0 0 0 0 0 0
Insert 8:	2	1	0 1 1 0 0 0
Insert 12:	0	3	1 1 1 1 0 0

Now let us attempt some membership queries:

	h1(x)	h2(x)	Answer
Query 14:	2	1	Yes (Wrong Answer 14 not inserted)
Query 10:	4	5	No (Correct Answer 10 not inserted)

In this example 8 and 12 are inserted into set A and a corresponding bloom filter B is set for set A. Due to these insertions first four bits of the bloom filter are set. When a person queries for number 14 in set A the answer he gets is yes i.e. the number is present. The reply to the query is yes because hashing functions generates index values 1 and 2 which were set in the bloom filter. So this results in a wrong answer to query. When query is made for number 10 the answer comes correct because index values 4 and 5 which were generated by counting bloom filter for number 10 were not set in the bloom filter.

This example shows that bloom filter has limitation and it does not always give correct answer to queries. This limitation can be reduced to a certain extent by using a variation of bloom filter which is known as counting bloom filter.

### 3.2 Counting Bloom Filters

A Counting Bloom Filter (denoted CBF) is a vector  $B$  of  $m$  **counters**. Along with it available to us are  $k$  (random hash) functions, each of which maps or hashes some set element to one of the  $m$  array positions.

To insert an element into a set the element is passed into  $k$  hashing functions and  $k$  index values are obtained. All counters in counting bloom filter at corresponding index values are incremented. To delete an element from the set reverse process is followed and corresponding counters are decremented.

Thus a counting Bloom filter (CBF) generalizes a Bloom filter data structure so as to allow membership queries on a set that can be changing dynamically via insertions and deletions. Fig.3 shows a Counting Bloom Filter.



Fig. 3 Counting Bloom Filter

### 3.3 Prevention Model

The prevention model we follow here was proposed in [2]. The prevention model is an instant detection model and it prevents the malicious transactions from getting committed. Malicious transactions are major threat to organizations nowadays. Majority of malicious transactions are performed by the people internal to organization. This model shown in Fig. 4 acts before the database and prevents the transactions from getting committed in database. To build the instant detector we use a counting bloom filter.

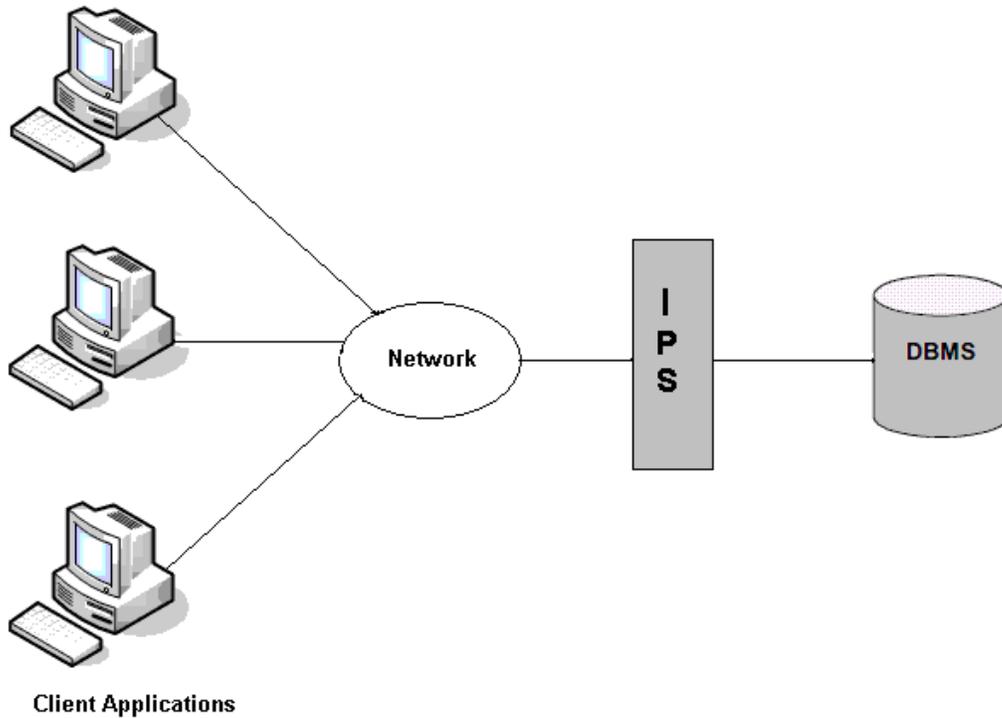


Fig.4 Database Management System with Intrusion Prevention System

This prevention model consists of Client Applications, Network and a database management system equipped with Intrusion Prevention System (IPS). The intrusion detection system in DBMTD mechanism proposed in [1] was acting after the DBMS as shown in Fig.1. Thus it allowed all the transactions to be committed before detecting whether it is malicious or not. Our intrusion prevention system comes in between the network and DBMS as shown in Fig. 4. Thus it checks all the transactions before they are committed and prevents the malicious ones from getting committed. So this model will even help us to avoid damage from those malicious transactions which cannot be rolled back. Unlike the DBMTD mechanism which fails to avoid damage in such a situation.

## 3.4 Intrusion Prevention System

Setting up an intrusion prevention system along with a database management system comprises of three phases namely (1) Profiling the transactions and assigning to users (2) Giving weight to commands and setting the CBF's for transactions and (3) Instant detection and prevention.

### 3.4.1 Profiling the transactions and assigning to users

Profiling consists of representing the authorized transactions as sequences of valid commands. In this phase the administrator and higher authorities sits together and identifies different transactions that are possible in the system. These transactions are then manually organized into a strict sequence of valid commands. Some typical transaction profiles are shown in fig. 5. Profiling of transactions also prevents attack of intruders to a certain extent because unless the intruder knows the strict sequence of commands he cannot perform a transaction.

An important aspect is that, the nodes in the graph do not represent concrete commands as commands may differ among executions. For example, consider the following SQL command to select the data from a given customer: *select name, address, phone from customer where name='John Carter'*. The name in the select criteria (*name=?*) depends on the target customer. This way, instead of considering concrete commands we have to represent those commands in a generic way. For example, the command to select data from a given customer can be represented by the following attributes: command type (*select*), target object (table *customer*), columns selected (*name, address, and phone*), and restriction field (*name*).

In addition to this the list of the users authorized to execute each transaction must also be defined. For instance, in the examples presented in Figure 5 User A is authorized to execute only transaction (a), User B is authorized to transaction (b) and User C is authorized to only transaction (c).

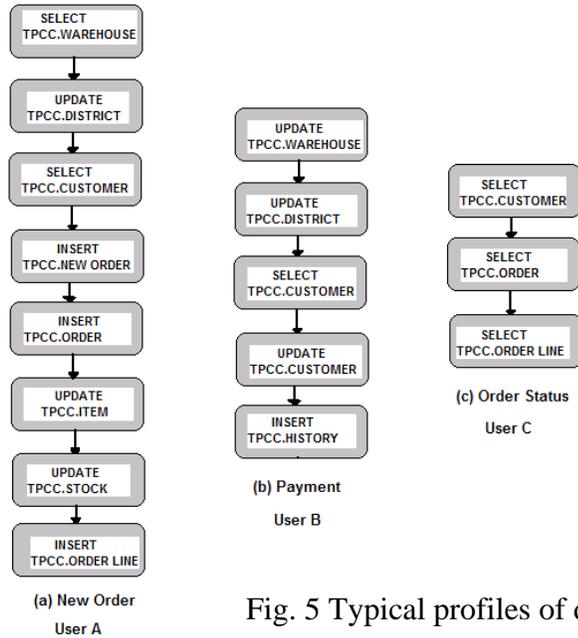


Fig. 5 Typical profiles of database transactions

### 3.4.2 Giving weight to commands and setting the CBF's for transactions

In this phase the DBA assigns weights to all commands identified during profiling phase. Weights can be assigned randomly. After this phase each transaction can be viewed as a strict sequence of weights. The list of commands and their corresponding weights is stored. After setting weights of all commands the next job of DBA is to set a counting bloom filter for each transaction. After setting all counting bloom filters they are also stored so that they can be loaded during the instant detection and prevention phase.

### 3.4.3 Instant Detection and Prevention

In this the user is allowed to enter commands but at any time the user can only execute one transaction. The same user performing transactions from one client machine cannot execute two transactions in parallel. When a user selects a particular transaction like making a payment or submitting a new order, etc IPS loads corresponding CBF and a weight\_command list. The weight\_command list contains weights and corresponding commands allowed in the system. If user's command is valid then counter values in CBF are decremented using weight of the identified command. The user is allowed to enter all the commands one by one. When user stops entering commands the counting bloom filter is checked. If all the counter values in CBF are zero then the transaction is declared as valid and committed into database otherwise the transaction is treated as a malicious transaction. This procedure is summarized in Instant\_Detector() algorithm.

Algorithm:

```
Instant_Detector (CBF[], weight_command[] )
```

```
// CBF[]: Counting Bloom filter set for the transaction started by user
```

```
// weight_command[]: weights of all commands possible and corresponding commands
```

```
{
```

```
Initialize user_entry[] ;
```

```
Initialize user_entry_weight ;
```

```
Do {
```

```
    Accept user input in user _entry[] ;
```

```
    If (user_entry[] is valid)
```

```
        // Check user_entry[] against all possible commands in weight_command[] list
```

```
        {
```

```
            Find weight of user_entry[] and assign it to user_entry_weight;
```

```
            Decrement CBF[] with corresponding hash functions and using user_entry_weight;
```

```
        }
```

```
    Else
```

```
        {
```

```
            Declare Transaction as invalid;
```

```
            Rollback Transaction;
```

```
            Exit Loop;
```

```
        }
```

```
    } while the user still wants to enter the values;
```

```
If any value in CBF is non zero
```

```
    Transaction is malicious, Rollback Transaction;
```

```
Else
```

```
    Commit the transaction;
```

```
}
```

Now we take an example to explain this algorithm. Consider a situation where a user logs in selects a particular transaction. As soon as transaction is selected a CBF[] will be loaded along with weight\_command list[]. Let the valid transaction has following sequence of weights {32,20,25,86,87}. Corresponding CBF is shown below.

4	1	3	0	2
---	---	---	---	---

Corresponding hashing function used are

$$H_1(x) = x \% 5$$

$$H_2(x) = (2x + 5) \% 5$$

Suppose the user enter correct sequence of commands then how Instant\_Detector() algorithm will identify it is shown below.

	$H_1(x)$	$H_2(x)$	CBF				
Initial CBF:			4	1	3	0	2
Use 32:	2	4	0	0	1	0	1
New CBF:			4	1	2	0	1
Use 20:	0	0	2	0	0	0	0
New CBF:			2	1	2	0	1
Use 25:	0	0	2	0	0	0	0
New CBF:			0	1	2	0	1
Use 86:	1	2	0	1	1	0	0
New CBF:			0	0	1	0	1
Use 87:	2	4	0	0	1	0	1
New CBF:			0	0	0	0	0

Finally after the user has entered all commands the value of all counters in CBF becomes zero. This shows that the transaction is a valid transaction and can be committed to the database.

# **Chapter 4**

## **Experimental Evaluation**

**Experimental Setup**

**Experimental Results and Discussion**

# Chapter 4

## 4 Experimental Evaluation

This section presents a practical example of the implementation of the Intrusion prevention system mechanism. In the experimental evaluation we have used the standard benchmark TPC-C [7] to generate profiles of valid transactions. Randomly generated transactions have been used to simulate malicious transactions.

### 4.1 Experimental Setup

The IPS mechanism implemented using Java in NetBeans IDE 6.1 is an autonomous application that runs separately from the DBMS.

TPC-C is based on a database with nine tables with several relationships and specifies five different types of transactions: entering orders (*new-order*), recording payments (*payment*), checking the status of orders (*order-status*), delivering orders (*delivery*), and monitoring the level of stock at the warehouses (*stock-level*). The TPC-C benchmark describes clearly the profile of these five types of transactions. Based on the TPC-C specification [7] we have manually built the profile of the transactions. Figure 6 presents those profiles.

Malicious transactions are submitted by an external application that connects to the DBMS using valid credentials (i.e., valid username, password, and user privileges). The malicious transactions are simulated by randomly generating transactions. An important aspect is that, the number of commands in each transaction ranges from 1 and 7. As the average number of commands in the valid TPC-C transactions is around four, we have decided to distributed the number of commands in the malicious transactions in the following way: 1) 40% of the malicious transactions contain four commands; 2) 30% of the transactions comprise three or five commands; 3) 20% of the transactions have two or six commands; and 4) 10% of the

transactions have 1 or 7 commands. It is worth mentioning that transactions with only one SQL commands and transactions with many commands (7 or more) are not frequent.

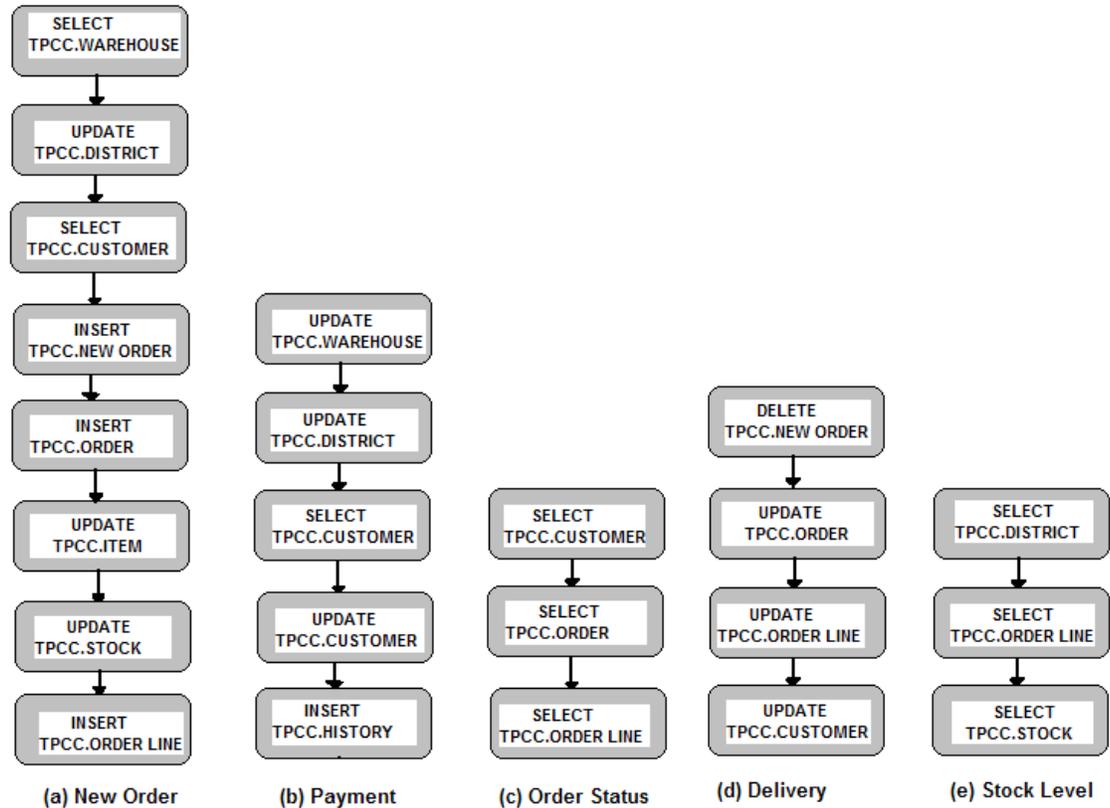


Fig.6 Profile of the TPC-C transactions

## 4.2 Experimental Results and discussion

The efficiency of IPS mechanism can be characterized by following measures: (1) **false negatives** (number of malicious transactions identified as valid) (2) **coverage** (percentage of successful malicious transactions detected).

### 4.2.1 False Negatives

False Negatives arises when IPS mechanism detects a malicious transaction as valid. Percentage of false negatives depends on the size of CBF used and number of hashing functions used. By

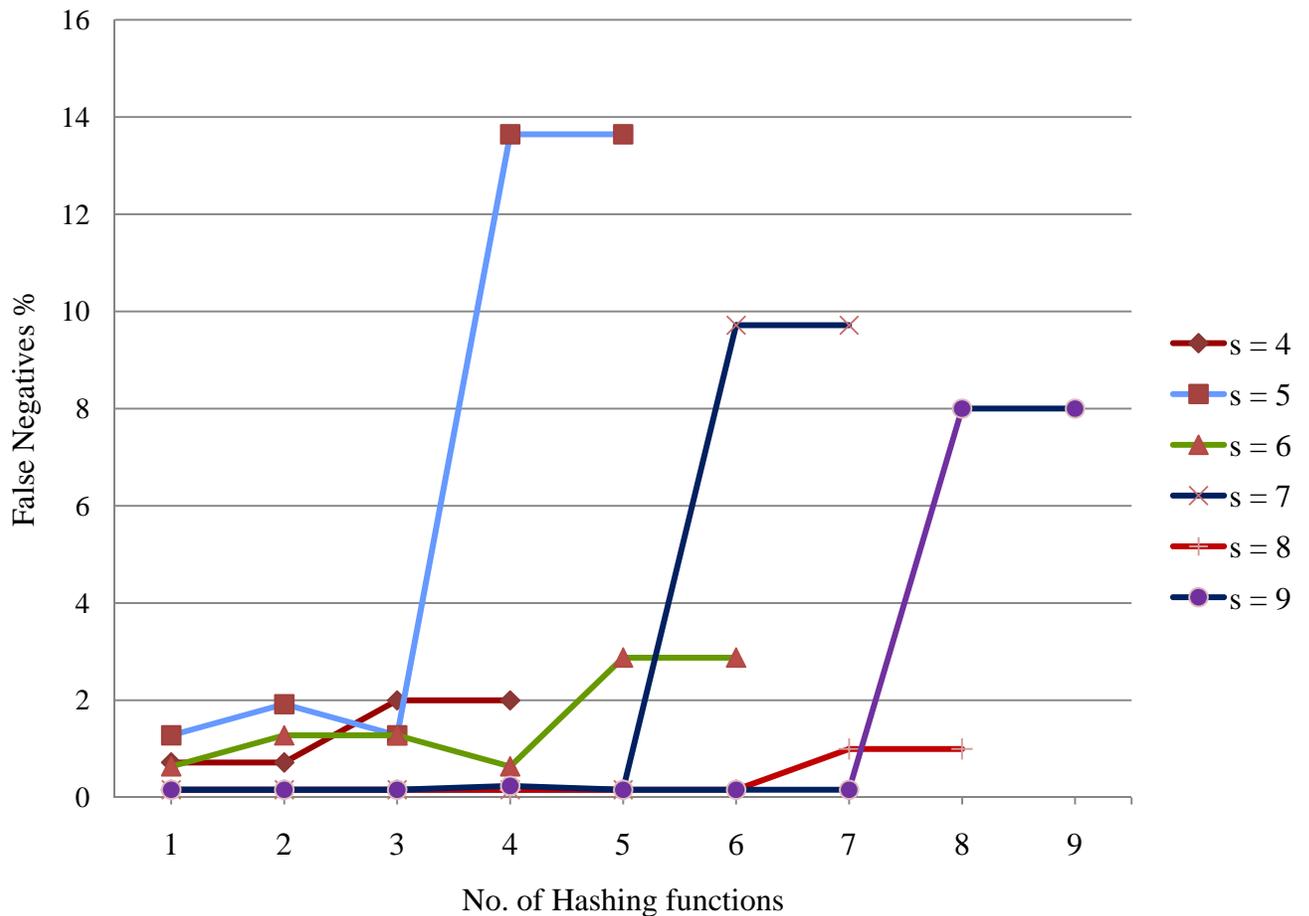
varying the size of CBF from 1 to 15 and number of hashing functions used from 1 to 15 different configurations of IPS mechanism are tested. For each configuration of IPS mechanism five transactions described in the TPC-C benchmark are selected as valid transactions. Based on TPC-C specification profile of the transactions is manually built as shown in Figure 6. Then all commands used in the transaction profiles are assigned random weights and corresponding CBF are set for each transaction. For each configuration of IPS mechanism 10000 randomly generated malicious transactions are submitted and results are recorded. The following section shows the graphs plotted using the results of the experiment.

### 4.2.1.1 Variation of false negatives percentage with increase in number of hashing functions

As the false negative percentage may depend on number of hashing functions used we decided to vary the number of hashing functions keeping the size of CBF constant to measure the impact of use of different number of hashing functions on false negatives percentage. Figure 7 presents the results.

Results in figure 7 shows that false negatives percentage highly depends on number of hashing functions used. False negatives percentage increases with increase in number of hashing functions for any size of CBF used if size of CBF used is kept constant. Results in figure 7 also shows a sudden rise in percentage of false negatives when size of CBF used is odd where as in case when size of CBF used is even there is a gradual increase in percentage of false negatives.

Fig. 7 variation of False Negatives % with increase in no. of hashing functions for different sizes of CBF (s)



#### 4.2.1.2 Variation of false negatives percentage with increase in size of CBF

As the false negative percentage may also depend on size of CBF used we decided to vary the size of CBF keeping number of hashing functions constant to measure the impact of use of different size of CBF on false negatives percentage. Figure 8 and Figure 9 presents the results.

Results in Figure 8 shows the variation in false negatives percentage when odd number of hashing functions is used. Results in figure 9 shows the variation in false negatives percentage when even number of hashing functions is used. It is seen that false negative percentage highly depends on size of CBF used. False negatives percentage decreases with increase in size of CBF used if number of hashing functions used is kept constant. This is seen for all number of hashing functions used. Figure 8 shows an exception that whenever size of CBF used equals number of hashing functions used + 1; there is an unacceptable rise in false negatives percentage. So such configurations of IPS mechanism must be avoided.

Fig . 8 variation of false negatives % with increase in size of CBF for different odd no. of hashing functions (hf)

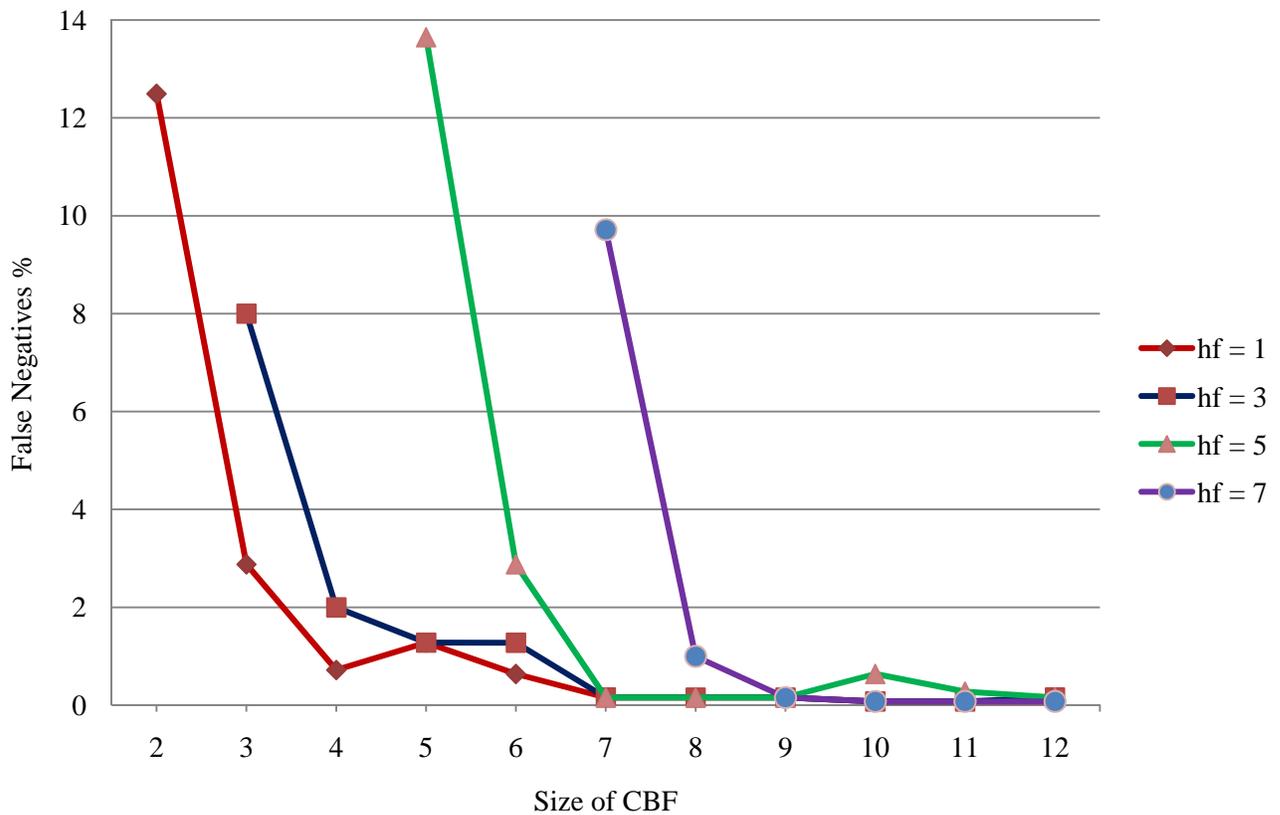
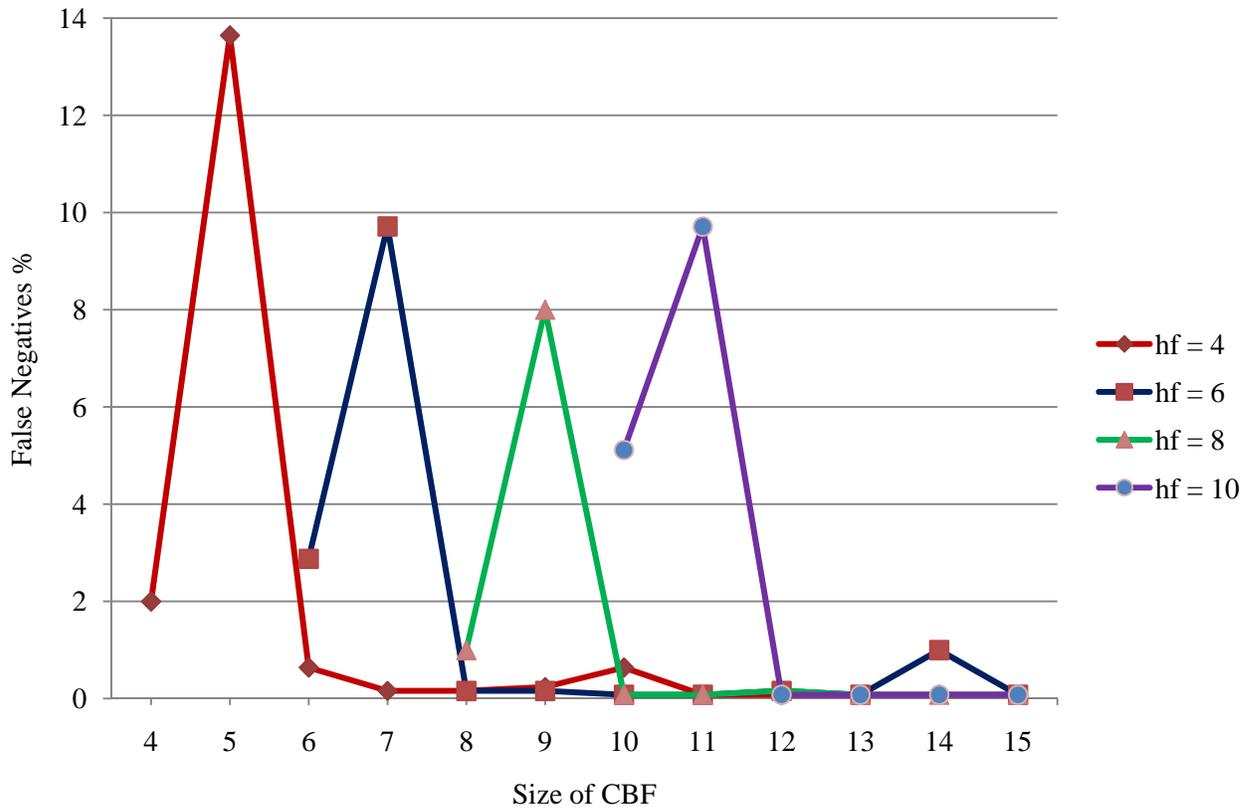
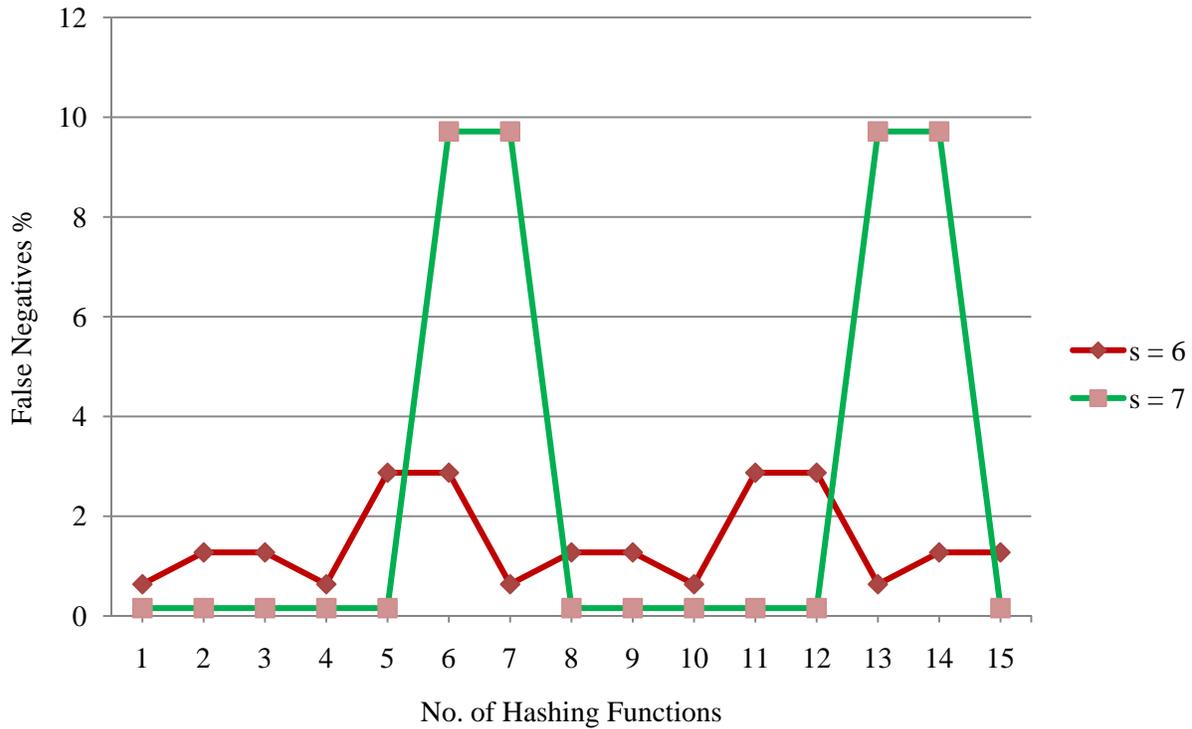


Fig . 9 variation of False Negatives% with increase in size of CBF for different even no. of hashing functions (hf)



Apart from these Figure 10 shows a repetitive pattern observed during the experiment. It appears that this repetitive pattern is observed because of the modulus operator used in hashing functions. From this pattern we can conclude that number of hashing function used should be less than or equal to size of CBF used. Using more number of hashing function will unnecessarily increase the computation time.

Fig. 10 Repetitive Patterns



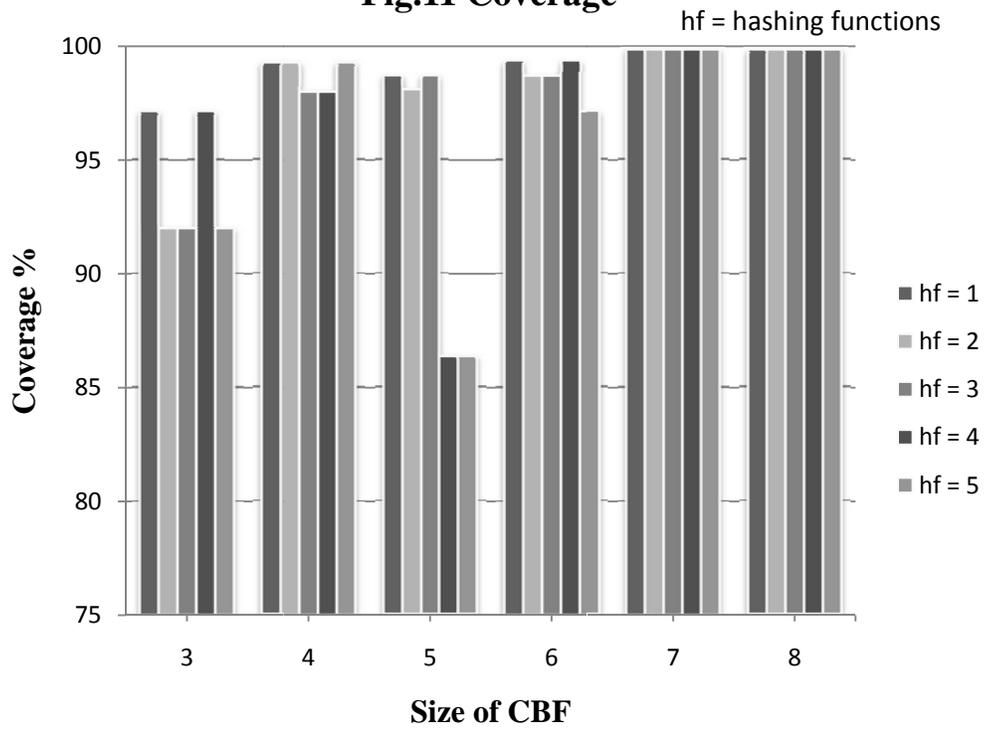
### 4.2.2 Coverage

The coverage represents number of malicious transactions detected. As already discussed before each pair of size of CBF and number of hashing functions used represents a different configuration of IPS mechanism. 10000 malicious transactions are submitted for every configuration of system and number of malicious transactions detected are recorded. Then Coverage percentage is calculated by using following formula

$$\text{Coverage\%} = ((\text{No. of malicious transactions detected}) * 100) / (\text{No. of malicious transactions submitted})$$

Figure 11 shows the coverage results. X axis corresponds to size of CBF and Y axis corresponds to coverage percentage. For every size of CBF coverage is plotted for five different number of hashing functions. Figure 11 show that for size of CBF greater than equal to 7; system can achieve coverage of about 99.85%, which is quite good result. Even for size of CBF equal to 6 the coverage is always greater than 95%.

**Fig.11 Coverage**



# **Chapter 5**

## **Conclusion and Future Work**

# Chapter 5

## 5 Conclusion and Future Work

This thesis proves that IPS mechanism can be used for detection of malicious transactions in DBMS. The IPS mechanism uses a counting bloom filter which is set for all the allowed transactions beforehand. The IPS mechanism comprises of three phases namely (1) Profiling the transactions and assigning to users (2) Giving weight to commands and setting the CBF's for transactions and (3) Instant detection and prevention.

The thesis presented a practical example of the implementation of IPS mechanism, which has been evaluated using a standard benchmark of database systems (TPC-C).The results show that 99.85% of the malicious transactions can be detected if best combination of size of CBF and number of hashing functions are used. The size of CBF greater than equal to 7 and only one or two hashing functions can be used to achieve this. This results show that IPS mechanism can be successfully applied to DBMS.

As future work learning methods capable of automatically identifying the profile of database transactions can be incorporated into IPS mechanism. Manual transactions profiling can be easily performed when the DBA knows the execution profile of the client database applications. On the other hand, automatic profiling does not require the DBA to know the profile of the allowed transactions. In the present version of IPS mechanism the profiles of the valid transactions are defined manually by the DBA.

## References

- [1] Marco Vieira and Henrique Madeira, "Detection of Malicious Transactions in DBMS", IEEE Proceedings- 11th Pacific Rim International Symposium on Dependable Computing, Dec 12-14, 2005, PP: 8.
- [2] Korra Sathya Babu, "Prevention of Malicious Transactions in DBMS", M.Tech Thesis, Department of computer Science and Engineering, NIT Rourkela, 2008.
- [3] E. F. Codd, "A Relational Model of Data for Large Shared Data Banks ", Comm. of the ACM (1970).
- [4] Ravi Sandhu and Pierangela Samarati, "Access Control: Principles and Practice", IEEE Communications Magazine, September 1994.
- [5] Yi Hu and Brajendra Panda, "Identification of malicious transactions in Database Systems", Proceedings of 7th International database engineering & Applications symposium, 16-18 July, 2003, PP 329-335.
- [6] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol", IEEE Transactions on Networking, 2000, PP 281-293.
- [7] TPC Council, "TPC Benchmark<sup>TM</sup> C, Standard Specification, Version 5.10.1", February 2009, <http://www.tpc.org/tpcc/>