Military Technical College
Kobry El-Kobbah,
Cairo, Egypt

7th International Conference
on Electrical Engineering
ICEENG 2010

Performance Evaluation of the Detection Mechanisms for Malicious Transactions in DBMS

By

Wasim Khalil Shalish * G. I. Salama ** H. A. Mohamed **

Abstract:

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 fact, several mechanisms needed to detect unauthorized database transactions executed by authorized or unauthorized users have been proposed and/or consolidated in the database arena. Most of these mechanisms can be implemented either externally as an autonomous subsystems separated from the DBMS (sharing the same machine or, preferably, in a dedicated machine), or implemented internally to the DBMS using database triggers. However, in the case of database triggers the performance degradation is expected to be quite high as the execution of database triggers is normally a high resource consuming task. In this paper, a proposed mechanism for the detection of unauthorized transactions in DBMS is implemented. The proposed mechanism implemented internally to the DBMS using database procedures by compiling them into native code residing in shared libraries. This paper presents a practical example of three mechanisms for detection of malicious transactions in DBMS, the proposed mechanism, internal database triggers, and external procedure. Finally, this paper investigates the performance of the three implemented malicious transactions detection mechanisms in the Oracle 10g DBMS and evaluates the mechanisms using a telephone database. The experimental results showed that the external procedure and the native mechanisms provide the greatest performance gains for computation-intensive procedural operations compared with the database triggers.

Keywords:

DBMS, Intrusion Detection System, Malicious Transactions.

* Syrian Armed Forces
** Egyptian Armed Forces

1. Introduction:

Due to the growth of networked data, security attacks have become a dominant practical problem all information infrastructures. Security violations are typically categorized as unauthorized data observation, incorrect data modification, and data unavailability. Unauthorized data observation results in the disclosure of information to users not entitled to gain access to such information. Incorrect modifications of data, either intentional or unintentional, result in an incorrect database state. Any use of incorrect data may result in heavy losses for the organization. When data is unavailable, information crucial for the proper functioning of the organization is not readily available when needed [1].

Thus, a complete solution to data security must meet the following three requirements [1]: 1) secrecy or confidentiality refers to the protection of data against unauthorized disclosure, 2) integrity refers to the prevention of unauthorized and improper data modification, and 3) availability refers to the prevention and recovery from hardware and software errors and from malicious data access denials making the database system unavailable. These three requirements arise in practically all application environments.

A typical database application is a client-server system (either a traditional client-server or a three tier 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.

A very important notion in DBMS is the concept of transaction [2]. In a simplified view, a transaction is a set of commands that performs a given action and takes the database from a certain consistent state to another consistent state. To guarantee that users only do what they are authorized to do, the DBMS normally implement a security mechanism based on user privileges. These privileges allow the system to control the actions that the users are allowed to perform. Two types of user's privileges are normally provided: system privileges and object privileges. System privileges are related to the actions that the users can perform (e.g., create users, create procedures, create tables, create indexes, etc.). Object privileges are related to the access to the database tables (e.g., selecting, deleting, updating, inserting, etc.)

For a posteriori analysis of the data accessing, allowing the detection of unauthorized accesses or anomalous usage, DBMS provide auditing mechanisms that record all the commands executed by the users (e.g., login, select table, insert into table, delete from table, etc.). The activation of the auditing mechanism is normally optional and the type of actions and the users audited are defined by the database administrator

(DBA). These actions are stored in a database table that can be used by the DBA to manually detect potential intrusion.
The structure of the paper is as follows: Section 2 introduces an overview of the mechanisms for malicious transactions detection and prevention in DBMS. The architecture of mechanisms for the implemented malicious transactions detection in DBMS is presented in section 3, section 4 presents the proposed mechanism, and Section 5 reports the experimental results to investigate the effects of the implemented malicious transaction detection mechanisms in a database performance using a telephone database, finally, section 6 concludes the paper and introduces future work.

2. Related Work:

Several mechanisms needed to protect data have been proposed and/or consolidated in the database arena [3, 4, 5]. Some examples include user authentication, user privileges, data encryption, auditing, etc. .
Liu and Heckman [3] presented the design of a real-time data attack isolation system, denoted by (DAIS). DAIS used triggers and transaction profiles to keep track of the items read and written by transactions, isolates attacks by rewriting user SQL statements, and is transparent to end users. Mattsson [4] offered the ability to detect misuse and subversion through the direct monitoring of database operations inside the database host, providing an important complement to host-based and network-based surveillance. Luenam and Liu [5] presented the design of an adaptive intrusion tolerant database system called (AITDB). AITDB was designed to enhance the cost effectiveness of ITDB through dynamic reconfiguration. Using a rule-based adaptation mechanism and a set of reconfiguration operators, AITDB automatically adapted itself to the dynamic changes of environment according to a specific set of adaptation (or tuning) criteria, which are determined based on the current situation of the damage (i.e., the data integrity level), the attacks, the workload, the availability level, the performance, and the cost.
However, most of these mechanisms are meant to place a barrier between the intruder and the data not to detect and handle the intrusion.
There has been some work on malicious transaction detection in DBMS [6, 7, 8], Vieira and Madeira [6] proposed a new mechanism for the detection of malicious transactions in DBMS and evaluates the mechanism using the TPC-C benchmark. Bockermann, et al [7] propose a novel approach for modeling SQL statements to apply machine learning techniques, such as clustering or outlier detection, in order to detect malicious behavior at the database transaction level. The approach incorporates the parse tree structure of SQL queries as characteristic e.g. for correlating SQL queries with applications and distinguishing benign and malicious queries. Kamra, et al [8] developed advanced security solutions for protecting the data residing in a DBMS. The strategy is to develop an Intrusion Detection (ID) mechanism, implemented within the database server that is capable of detecting

anomalous user requests to a DBMS. The key idea is to learn profiles of users and applications interacting with a database.

3. The Architecture of a Database System with the Implemented Malicious Transactions Detection Mechanisms:

The mechanisms for the detection of malicious transactions in DBMS are implemented externally as an autonomous subsystems separated from the DBMS, internally to the DBMS using database triggers, and internally to the DBMS using database procedures by compiling them into native code residing in shared libraries. Figure 1 presents the basic architecture of a database system with the malicious transactions detection mechanism using external procedure.

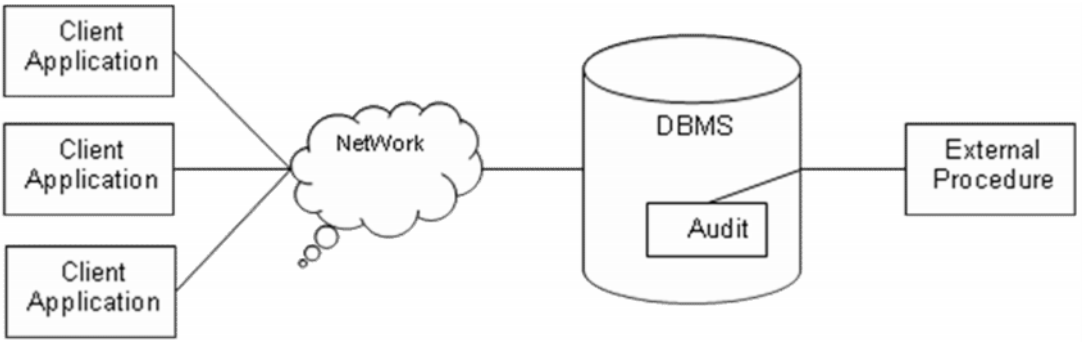

Figure (1): *The architecture of a database system with the malicious transactions detection mechanism using the external procedure.*

Figure 2 presents the basic architecture of a database system with the malicious transactions detection mechanism using database triggers or native code.

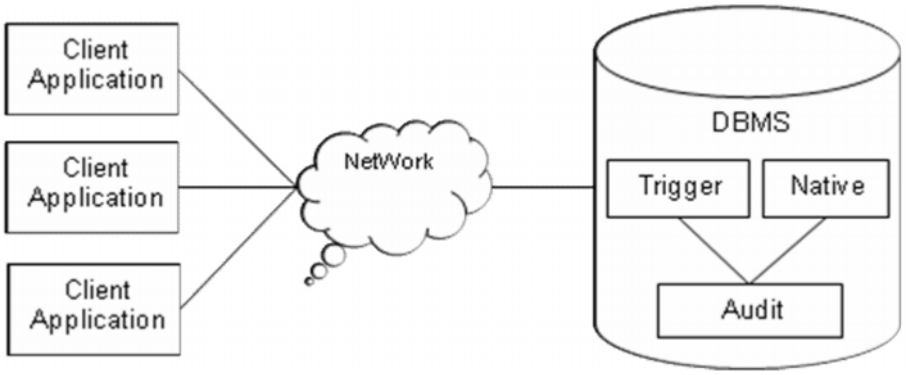

Figure (2): *The architecture of a database system with the malicious transactions detection mechanism using database triggers or native code*

As shown in figure 1 and 2, the malicious transactions detection mechanism is built on top of the auditing mechanism implemented by most commercial DBMS.

Implementation of the malicious transaction detection mechanisms externally lets you [9]:

1) Isolate execution of client applications and processes from the database instance to ensure that any problems on the client side do not adversely impact the database.
2) Move computation-bound programs from client to server where they execute faster (because they avoid the round-trips of network communication).
3) Interface the database server with external systems and data sources.
4) Extend the functionality of the database server itself.

Although, implementation of the mechanisms externally has provided a number of features that make them truly an industrial-strength resource, there are some limitations of these mechanisms [9]:

1. Despite the wonders of shared libraries, database architecture requires an unavoidable amount of inter process communication.
2. The overhead for the first external procedure call from a given session may result in a noticeable delay in response time; however, subsequent calls are much faster.

On the other hand, implementation of the mechanisms internally using database triggers has several benefits [10]:

1. Database triggers are very efficient in a client-server environment, triggers modules fire automatically, independent of the calling process, thus cutting down on client-to-server network communication and assisting in the laborious task of software distribution.
2. Provide enhanced and complex security checks an auditing.
3. Improve data integrity.

However, database triggers have a number of disadvantages. Here we discuss the most important ones [10]:

1. Unlike stored procedures and packages, triggers are not held in the database in a compiled (PL/SQL pcode) form. Every time a trigger is fired, the code must be recompiled.
2. When a database update fails, the offending statement and all previous trigger updates are also rolled back.
3. Database triggers can be accidentally disabled or dropped by a person with sufficient privilege. This is actually nothing new. The problem is that there is no practical way of writing application code to guarantee that a trigger actually exists. If your application uses database triggers to audit sensitive data or accumulate summary tables, a disabled trigger can compromise the integrity of your whole system.
4. The execution of database triggers is normally a high resource consuming task; in this case the performance degradation is expected to be quite high [11, 12].

Native compilation provides the greatest performance gains for computation-intensive procedural operations. Examples of such operations are data warehouse applications, and applications with extensive server-side transformations of data for display. In such cases, expect speed increases of up to 30%. If you do not use native

compilation, each program unit is compiled into an intermediate form, machine-readable code (m-code). The m-code is stored in the database dictionary and interpreted at run time. With native compilation, the program statements are turned into C code that bypasses all the runtime interpretation, giving faster runtime performance [13]. Limitation of the native compilation is that native compilation takes longer than interpreted mode compilation. That's because native compilation involves several extra steps: generating C code from the initial output of the compilation, writing this to the file system, invoking and running the C compiler, and linking the resulting object code into Oracle.

4. The Proposed Mechanism:

In fact, Oracle is transaction oriented. That is, Oracle uses transactions to ensure data integrity. A transaction is a series of SQL data manipulation statements that does a logical unit of work. For example, two UPDATE statements might credit one bank account and debit another. It is important not to allow one operation to succeed while the other fails [13].

Malicious database transactions can be classified as transactions executed by authorized users that use their normal privileges to execute illicit actions for their personal benefit or transactions executed by unauthorized users that gain access to the database by exploring system vulnerabilities.

As an example, we consider a telephone database where two employees have access to the secret database information about its subscribers. One of the telephone's employees is Scott, and the other is Ha. The following tables are used in this paper:

- Subscriber table, which contain details about subscribers
- City table, which contain details about cities
- Area table, which contain details about areas
- Services table, which contain details about services
- Telephone table, which contain details about numbers of telephone
- Bill table, which contain details about bills

The proposed mechanism consists of two different phases: transaction profiling corresponds to the identification of the sequence of commands that constitute each valid transaction, and malicious detection consists in the detection of users executing sequences of commands that potentially represent intrusion attempts.

4.1 Transaction profiling:

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 (authorized transactions) to identify user's attempts to execute invalid sequences of commands (malicious transactions). The DBA creates that profile as the DBA knows the execution profile of the client database applications.

A database transaction can be represented as a direct graph describing the different execution paths (sequences of selects, inserts, updates, and deletes) between the beginning of the transaction and the commit or rollback command.
Figure 3 shows examples of graphs representing the profile of database transactions. The nodes in the graph represent commands and the arcs represent the valid execution sequences.

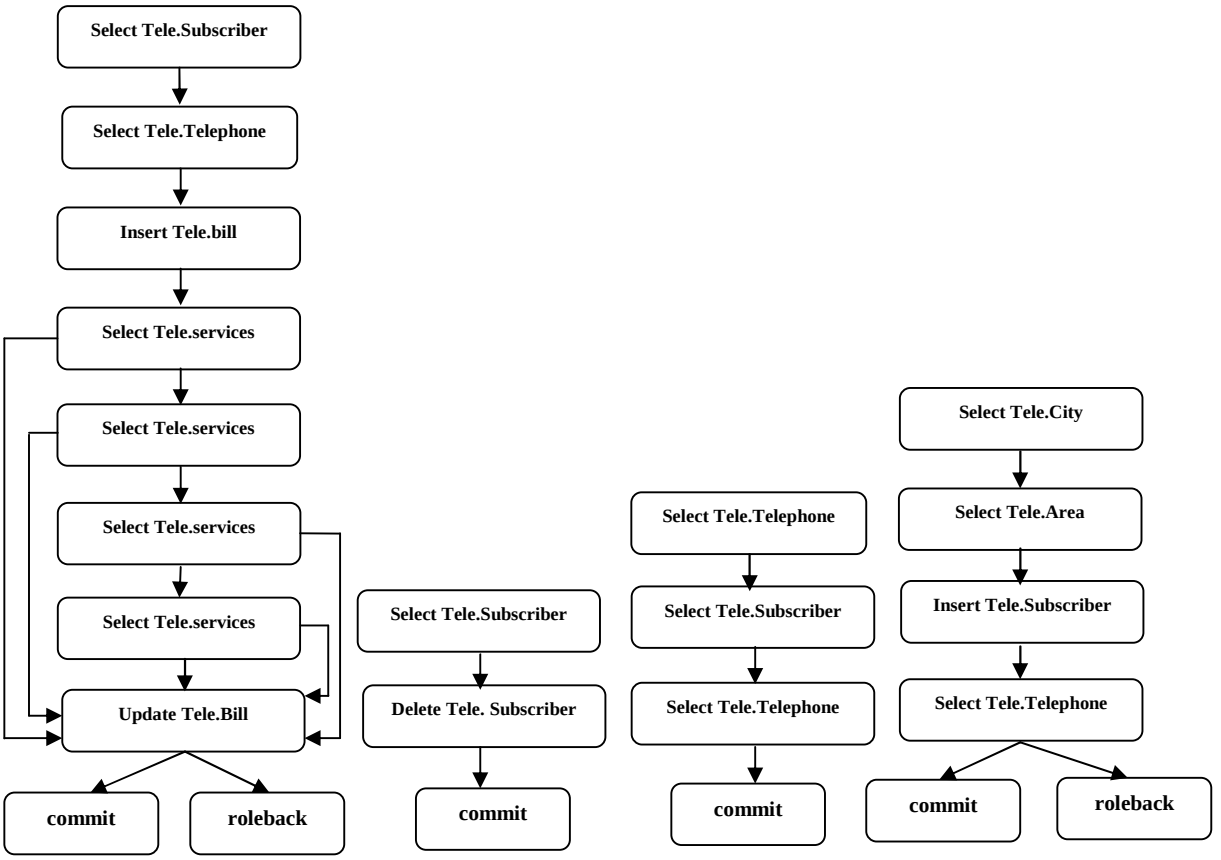

***Figure (3)**: Examples of typical profiles of database transactions*

For the considered telephone database there are four authorized transactions:

- Transaction 1: a set of eight commands represent a pay telephone bill
- Transaction 2: a set of two commands represent delete subscriber
- Transaction 3: a set of three commands represent information about subscriber
- Transaction 4: a set of four commands represent adding subscriber

We consider that employee Scott is authorized to execute transaction number (1, 2), while employee Ha is authorized to execute transaction number (3, 4).

4.2 Malicious detection:

The auditing mechanism is the most comprehensive and complete auditing facility within the Oracle database. It allows you to audit activities based on activity type, object, privilege, or user. The auditing mechanism collects information about the commands (e.g., login, select table, insert into table, delete from table, etc.) executed by the database users. This log is used by the proposed mechanism to obtain the sequence of commands recently executed by each user, which is then compared to the profile of the authorized transactions to identify potential malicious transactions.
The minimum information needed to characterize each command includes: username, identification of user session, sequential number that univocally defines the sequence of the command in the session, type of the command executed, name of the object targeted by the command, and identification of the user that owns that object. Table 1 presents an excerpt of an audit table
containing the minimum information required to implement the mechanism.

***Table (1)**: Excerpt of an audit table*

User name	Session id	Seq #	Command type	Target object	Object owner
scott	13206	9	select	Subscriber	telephone
scott	13206	10	select	Telephone	telephone
scott	13206	1	select	Bill	telephone
HA	13185	2	select	Subscriber	telephone
HA	13185	3	insert	Subscriber	telephone
scott	13225	5	Update	bill	PB
scott	13225	6	Insert	Bill	telephone

5. Discussion and Experimental Results:

The efficiency of the proposed mechanism can be characterized by its impact on database performance. In this section we present and discuss the results obtained for this measure. Finally, we investigate the performance of the three implemented malicious transactions detection mechanisms in the Oracle 10g DBMS.

5.1 Evaluation of the Proposed Mechanism:

Understanding the impact that the proposed and the auditing mechanisms have on the database performance is one of the most important things, especially if you have a high-volume environment and you have stringent auditing requirements that include auditing activities that happen a lot. An important note is that the auditing

mechanism affects performance and that the impact is directly proportional to how much you audit [14].

Three configurations have been considered in the evaluation of the impact on the database performance: 1) baseline (Oracle 10g fully tuned for performance and without using the audit mechanism); 2) audit (Oracle 10g using the audit mechanism but without malicious transactions detection); and 3) the proposed mechanism (Oracle 10g using audit with the malicious transactions detection mechanism).

As the impact in the performance caused by the audit and the proposed mechanisms may depend on the transactions submitted to the database, we have decided to measure the performance under the considered four authorized transactions. Figure 4 presents the results for the three configurations considered. The Y axis corresponds to the number of transactions executed per minute (tpm), and the X axis represents the four authorized transactions (T1 to T4).

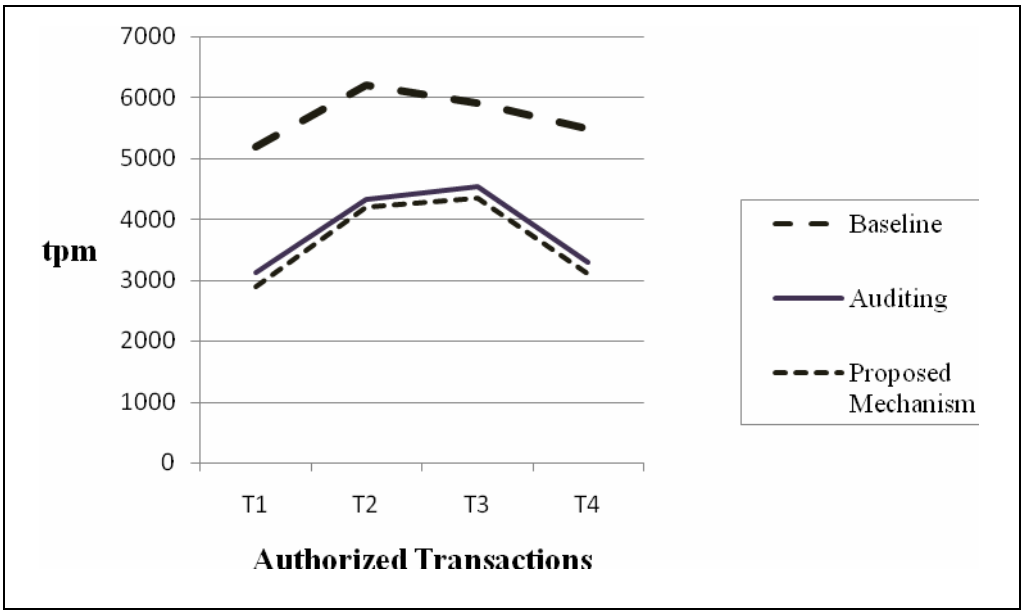

Figure (4): *Performance for the three configurations*

Results in Figure 4 show that the impact in the performance is highly related to how much you audit. As shown, the number of transactions executed per minute for the audit and the proposed mechanism configurations is lower than for the baseline configuration, especially for transactions T1 and T4.

From the analyses of the results presented in Figure 4 we can derive the performance overhead introduced by the standard auditing mechanism (without malicious transactions detection) and by the proposed mechanism. As we can see, the performance overhead caused by the activation of the auditing mechanism is fairly high (more than 40%) when transactions T1 and T4 are submitted. On the other hand, for transactions T2 and T3 the performance overhead varies between 30% to 33%. A very interesting result is that the performance penalty added by the proposed

mechanism is quite small (less than 5% independently of the transaction submitted). In other words, the performance penalty is due to the database audit mechanism.

5.2 Comparison of the Mechanisms:

In order to validate the proposed mechanism, we conduct extensive performance evaluation on the proposed mechanism on various simulated malicious transactions. The evaluation has been based on the comparisons with the external mechanism and internal database triggers in terms of the latency (time between the execution of the malicious transaction and its detection).
Different behavior concerning database performance and functionality are to be expected of the different mechanisms considered. Figure 5 shows the results obtained for latency time of the implemented mechanisms. An important aspect is that, the number of commands in each transaction ranges from 1 to 8.

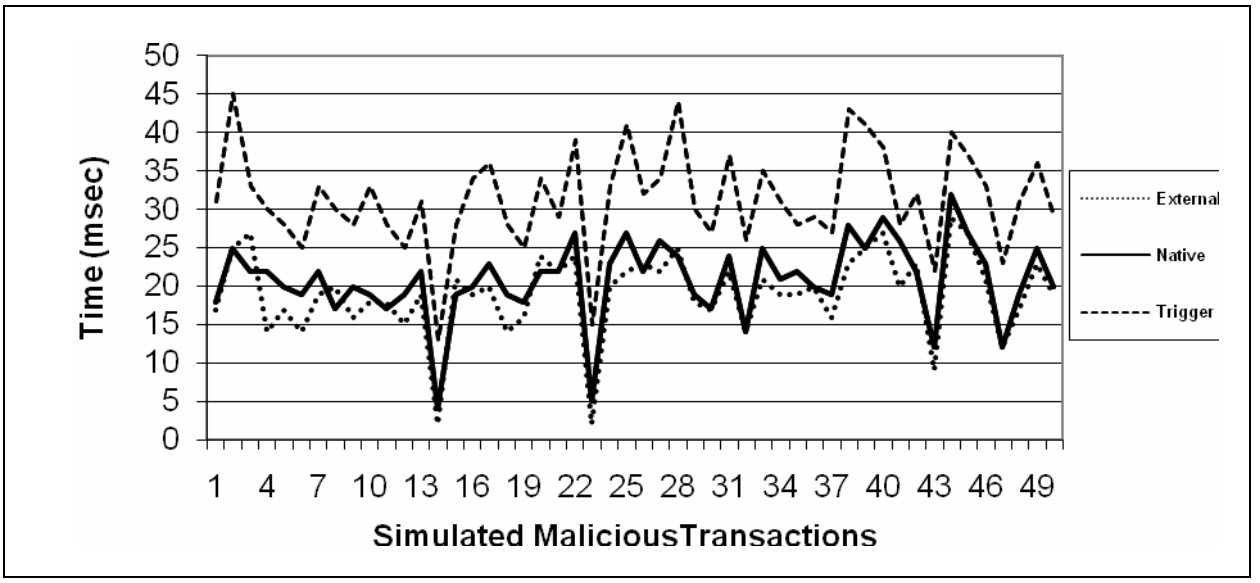

***Figure (5)**: Latency of the three mechanisms.*

From Figure 5, it could be noticed that for all the simulated malicious transactions, the latency of the database trigger mechanism is more than the latency of both the external and the native mechanisms. Also, it could be seen that the latency varies between 2 msec and 29 msec for the external mechanism, 4 msec and 32 msec for the native mechanism, and 13 msec and 45 msec for the database trigger mechanism. As expected, the native mechanism outperforms the database trigger mechanism in term of database performance, the average latency for the native mechanism is about 20 msec and about 31 msec for the database trigger mechanism, this explains the expected better performance of the native compilation over the database trigger mechanism.

Furthermore, the database performance is almost the same in case of external mechanism and native compilation, the average latency for the external mechanism is about 19 msec. Moreover, for all the considered transactions, the latency varies as the number of commands in each transaction varies, the higher the number of commands the higher the latency.

6. Conclusion:

This paper proposed a mechanism for the detection of malicious transactions in DBMS. The proposed mechanism implemented internally to the DBMS using database procedures by compiling them into native code residing in shared libraries. The proposed mechanism uses a graph that represents the profile of valid transactions to detect unauthorized transactions and consists of two different phases: transaction profiling and malicious detection.

The experimental results showed that the performance overhead caused by the activation of the auditing mechanism varies between 30% to 40%, and the performance penalty added by the proposed mechanism is quite small (less than 5% independently of the transaction submitted). In other words, the use of the proposed mechanism does not represent a significant performance problem as the performance penalty is due to the database audit mechanism.

Finally, this paper presents a practical example of implementation of three mechanisms for the detection of malicious transactions in DBMS, the proposed mechanism, internal database triggers, and external procedure for the Oracle 10g DBMS, which have been evaluated using a telephone database.

The results show that the malicious transaction detection mechanisms can be implemented externally or internally to the DBMS using the native code, as the external and the native mechanisms outperform the database trigger mechanism in term of database performance.

References:

[1] E. Bertino, R. Sandhu, *Database Security Concepts, Approaches, and Challenges*, IEEE Transactions on Dependable and Secure Computing, Vol. 2, No. 1, 2005.

[2] J. Gray, A. Reuter, *Transaction Processing: Concepts and Techniques*, The Morgan Kaufmann Series in Data Management Systems, 1993.

[3] P. Liu, *DAIS: A Real-time Data Attack Isolation System for Commercial Database Applications*, Journal of Network and Computer Applications, Volume 29, No. 4, 2006.

[4] U. Mattsson, *A Real-time Intrusion Prevention System for Enterprise Databases*, in Proceedings of the 4th WSEAS International Conference on Artificial Intelligence Knowledge Engineering Data Bases, Austria, 2005.

[5] P. Luenam, P. Liu, *The Design of an Adaptive Intrusion Tolerant*

Database System, International Conference on Foundations of Intrusion Tolerant Systems, p. 14-21, 2003.

[6] M. Vieira, H. Madeira, *Detection of Malicious Transactions in DBMS*, IEEE Computer Society in Proceedings of 11th Pacific Rim International Symposium on Dependable Computing, p. 350-357, 2005.

[7] C. Bockermann, M. Apel, and M. Meier, *Learning SQL for Database Intrusion Detection using Context-Sensitive Modeling*, in: Detection of Intrusions and Malware and Vulnerability Assessment, volume 5587, p. 196-205, Springer Berlin / Heidelberg, 2009.

[8] A. Kamra, E. Bertino, and G. Lebanon, *Mechanisms for Database Intrusion Detection and Response*, International Conference on Management of Data: Data security & privacy, p. 31-36, 2008.

[9] D. Adams, M. Cowan, R. Moran, J. Melnick, E. Paapanen, J. Russell, R. Strohm, handbook of *Oracle Database Application Developer's Guide – Fundamentals*, November 2005.

[10] S. Krishna, handbook of *Introduction to Oracle 9i: PL/SQL*, Student Guide. Vol. 2, June 2001.

[11] R. Ramakrishnan, J. Gehrke, *Database Management Systems*, 3rd ed., McGraw Hill, 2002.

[12] Oracle Corporation, *Oracle® Database Concepts 10g Release 1 (10.1)*, 2003.

[13] S. Agrawal, C. Barclay, E. Belden, and others, handbook of *Oracle Database PL/SQL User's Guide and Reference 10g Release 2 (10.2)*, June 2005.

[14] R.B.Natan, handbook of How to Secure and Audit Oracle 10g and 11g , Taylor & Francis Group, 2009.