Paper: ASAT-14-179-CE

14th International Conference on
AEROSPACE SCIENCES & AVIATION TECHNOLOGY,
ASAT - 14 – May 24 - 26, 2011, Email: asat@mtc.edu.eg
Military Technical College, Kobry Elkobbah, Cairo, Egypt
Tel: +(202) 24025292 –24036138, Fax: +(202) 22621908

Database Performance Sensitivity for Malicious Transactions Detection Mechanisms

W.K. Shalish*, A.Z. Ghalwash†, H.M. El-Deeb‡, G.I. Salama§, H.A. Mohamed**, K. Badran††

Abstract: Data represent today an important asset for companies and organizations and must be protected. Most of an organization's sensitive and proprietary data resides in a Database Management System (DBMS). Various attacks (e.g., malicious transactions) may corrupt data items in the database systems, which decreases the integrity level of the database. Malicious transactions detection mechanisms are becoming more and more sophisticated to detect such attacks. These mechanisms are implemented either externally as an autonomous subsystems separated from the DBMS, internally to the DBMS using database triggers, or internally to the DBMS using database stored procedures by compiling them into native code residing in shared libraries. The focus of this paper is to investigate the effect of malicious transactions detection mechanisms implementation on database performance. This paper presents implementation of three mechanisms for detection of malicious transactions in the Oracle 10g DBMS, and investigates the performance of the three mechanisms using a telephone database. The experimental results showed that the average performance overhead caused by the activation of the external and the native mechanisms is about 40%, and about 48% for the database trigger mechanism. As a result, the external and the native mechanisms outperform the database trigger mechanism in term of database performance.

Keywords: DBMS, Malicious Transactions.

1. Preface

As organizations increase their adoption of database systems as the key data management technology for day-to-day operations and decision making, the security of data managed by these systems becomes crucial. Damage and misuse of data affect not only a single user or application, but also may have disastrous consequences on the entire organization [1].
classified as three categories the first is intentional unauthorized attempts to access or destroy private data; second is 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 third is external interferences aimed to cause
Various attacks (e.g., malicious transactions) may corrupt data items in the database systems, which decreases the integrity level of the database. Typical database security attacks can be undue delays in accessing or using data, or even denial of service. In the present work we are

* Syrian Armed Forces (M.T.C), wseemgd@yahoo.com.
† Helwan University, atef_ghalwash@yahoo.com
‡ Modern University for Technology and Information (M.T.I), hmeldeeb14@yahoo.com
§ Egyptian Armed Forces, goda80@yahoo.com
** Egyptian Armed Forces, h_aboelsoud@yahoo.com
†† Egyptian Armed Forces, khaledBadran@hotmail.com

particularly interested in the first two types of attacks, which in practice corresponding to malicious transactions executed by authorized users or by unauthorized users that gain access to the database by exploring system vulnerabilities.

Several mechanisms needed to protect data have been proposed and/or consolidated in the database arena [2, 3, 4], Most of These mechanisms are implemented either externally as an autonomous subsystems separated from the DBMS, internally to the DBMS using database triggers, or internally to the DBMS using database procedures by compiling them into native code residing in shared libraries.

Implementation of the malicious transaction detection mechanisms externally lets you 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, move computation-bound programs from client to server where they execute faster (because they avoid the round-trips of network communication), interface the database server with external systems and data sources, and extend the functionality of the database server itself [5]. 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: despite the wonders of shared libraries, database architecture requires an unavoidable amount of inter process communication, and 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, implementations of the mechanisms internally using database triggers provides enhanced and complex security checks and auditing, and improve data integrity [6]. However, database triggers have a number of disadvantages: the execution of database triggers is normally a high resource consuming task; in this case the performance degradation is expected to be quite high [7, 8]. 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, and when a database update fails, the offending statement and all previous trigger updates are also rolled back.

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 [9]. 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 language code from the initial output of the compilation, writing this to the file system, invoking and running the C language compiler, and linking the resulting object code into Oracle.

In this work, we investigate the effect of malicious transactions detection mechanisms implementation on database performance. This paper presents a practical example of the implementation of the three mechanisms in the Oracle 10g DBMS and evaluates the mechanisms using a telephone database.

The structure of the paper is as follows: Section 2 introduces a group of published work in the area of malicious transaction detection mechanisms in DBMS. The implementation of malicious transactions detection mechanisms is presented in section 3, and Section 4 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 5 concludes the work and introduces future work for further investigation.

2. Published Work

Many researchers have dwelled into the field of performance evaluation of the malicious transaction detection mechanisms in DBMS [10, 11], Shalish, et al [10] proposed a mechanism for the detection of unauthorized transactions in DBMS. The proposed mechanism implemented internally to the DBMS using database procedures by compiling them into native code residing in shared libraries. They evaluated the mechanism by its impact in the database performance (performance overhead introduced by the mechanism). The experimental results showed that the performance penalty added by the proposed mechanism is quite small.

Ayushi, et al [11] proposed a new mechanism for the detection of malicious transactions in DBMS, the Database Malicious Transactions Detector (DBMTD) mechanism. The DBMTD mechanism is an autonomous application that runs separately from the DBMS in a dedicated machine. The published work presented a practical example of the implementation of the proposed mechanism for the Oracle 10g DBMS, which has been evaluated using a standard benchmark for database systems (TPC-C), The results show that more than 99% of randomly generated transactions (simulating malicious transactions) can be detected and subsequently rolled back while the performance penalty in normal conditions is less than 10% and the latency lower than 2 seconds. These results clearly show that malicious database transactions detection can be successfully applied to DBMS.

3. Malicious Transactions Detection Mechanisms Implementation

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

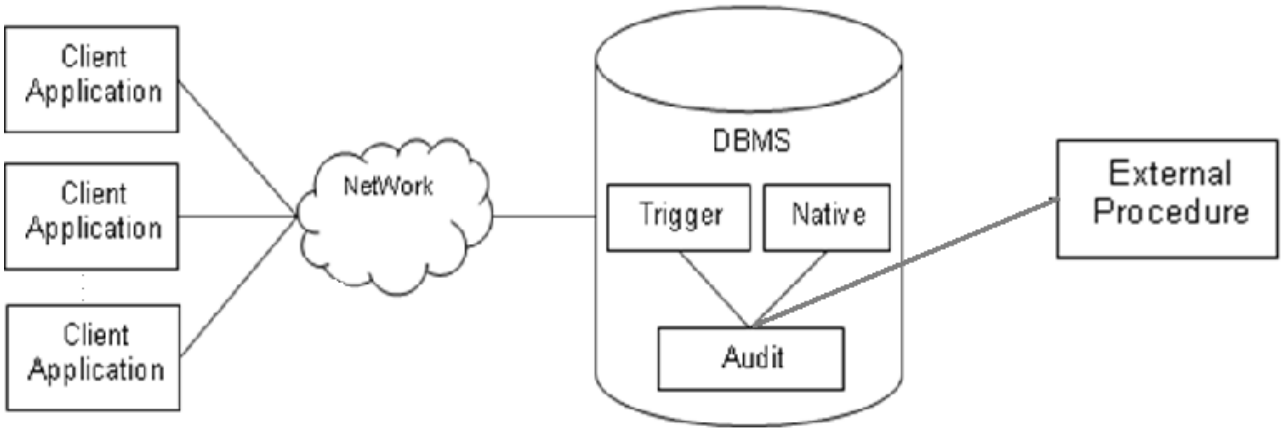

Fig.1. Architecture of a database system with the malicious transactions detection mechanism using the external, native, and trigger procedures

As shown in this figure the malicious transactions detection mechanisms are built on top of the auditing mechanism implemented by most commercial DBMS.

3.1 Implementation of MTDM using External Procedure

External procedures allow you to call anything that you can compile into the shared library format of the operating system. External procedures are reliable, multi-threaded, bidirectional communication and can be used as user-defined functions in SQL.
You can write the external routine in any language you wish, as long as your compiler and linker will generate the appropriate shared library format that is callable from C language. In Oracle, however C language will be the most common language for external procedures, since all of Oracle's support libraries are written in C language. Although Oracle has named these features of the external procedures, you are technically invoking C functions (same as invoking any procedure in C language). If the C function returns a value, you map it to a PL/SQL function; if it returns no value, you map it to a PL/SQL procedure [12].

Figure 2 shows the steps of invoking an external procedure. As shown in the diagram below, the process flow starts with a PL/SQL application that calls a special PL/SQL module body. PL/SQL then looks for a special Net8 listener process, which should already be running in the background. At this point, the listener will spawn an executable program called extproc. This process loads the dynamic library and then invokes the desired routine in the shared library, whereupon it returns its results back to PL/SQL.

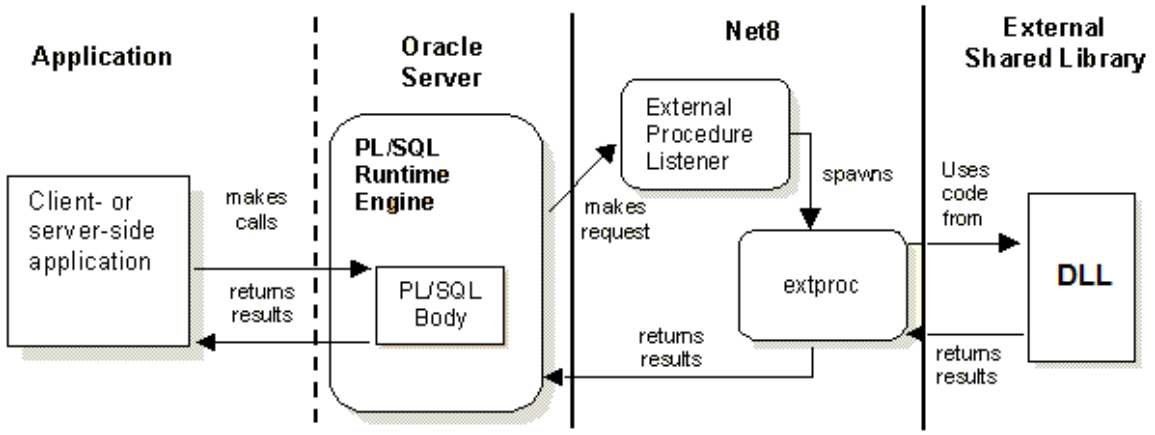

Fig.2. Steps of invoking an external procedure

3.2 Implementation of MTDM using Internal Triggers

Triggers are similar to stored procedures. A trigger stored in the database can include SQL and PL/SQL or Java statements to run as a unit and can invoke stored procedures. However, procedures and triggers differ in the way that they are invoked. A procedure is explicitly run by a user, application, or trigger. Triggers are implicitly fired by Oracle Database when a triggering event occurs, no matter which user is connected or which application is being used.
Fig.3 shows a database application with some SQL statements that implicitly fire several triggers stored in the database. Notice that the database stores triggers separately from their associated tables.

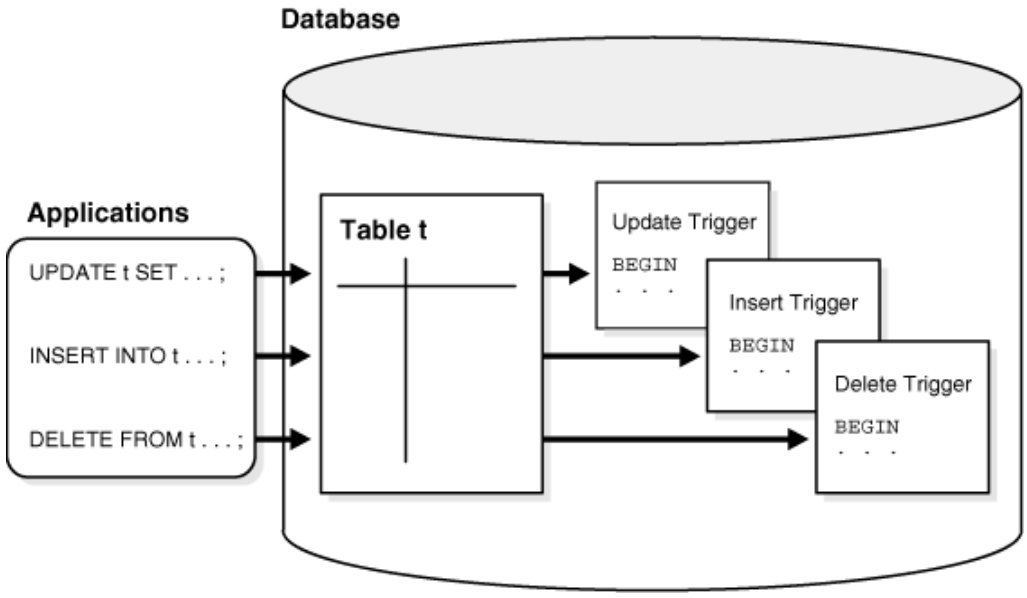

Fig.3. SQL statements that fire several triggers

3.3 Implementation of the MTDM using Internal Native Procedure

Native PL/SQL compilation is achieved by translating the PL/SQL source code into C code that is then compiled on the given platform. The compiling and linking of the generated C code is done using third-party utilities whose location has been specified by the DBA [5]. PL/SQL uses the supported operating system C compiler and linker, to compile and link the resulting C code into shared libraries. The shared libraries are stored inside the data dictionary, so that they can be backed up automatically and are protected from being deleted. These shared library files are copied to the file system and are loaded and run when the PL/SQL subprogram is invoked. If the files are deleted from the file system while the database is shut down, or if you change the directory that holds the libraries, they are extracted again automatically.

Although PL/SQL program units, that just call SQL statements might see little or no speedup, natively compiled PL/SQL, is always at least as fast as the corresponding interpreted code. The compiled code makes the same library calls as the interpreted code would, so its behavior is exactly the same, some of the setup steps require DBA authority. You must change the values of some initialization parameters, and create a new directory on the database server, preferably near the data files for instance. The database server also needs a C compiler, on a cluster; the compiler is needed on each node. Even if you can test out these steps yourself on a development machine, you will generally need to consult with a DBA and enlist their help to use native compilation on a production server.

4. Discussion and Experimental Results

4.1 Experimental Setup

The Oracle™ DBMS is one of the leading databases in the market and as one of the most complete and complex database. It represents very well all the sophisticated DBMS available today. For that reason, we have chosen the Oracle 10g DBMS [8] as a case study.

The telephone database is designed to simulate telecommunication centrals, and based on a database with 11^{th} tables with several relationships and specifies four different types of transactions: T1 (a set of **eight** commands represents a telephone bill payment), T2 (a set of **two** commands represents delete subscriber), T3 (a set of **three** commands represents

information about subscriber), and T4 (a set of **four** commands represents adding subscriber), otherwise it is considered to be malicious transaction. The telecommunication centrals database requirements are as follow: keeps track of telephone's subscribers, telephone's information (telephone entities), telephone's bills, telephone's services, telephone's complaints and faults, telephone's calls, telephone's area, city and usage. In this case, telecommunication central's database contains eleven entities: Subscriber, Telephone, Call, Bills, ComplaintFaultes, ComplaintType, TelephoneService, Service, Area, City, and UsageType.

4.2 Performance Evaluation Measures for the MTDMs

The efficiency of the MTDMs mechanisms can be characterized by the following measures: Impact on the database performance (performance overhead introduced by the mechanism)

$$\text{Impact\%} = \frac{(\text{tpm(auditing)} - \text{tpm(MTDM)}) * 100}{\text{tpm(auditing)}} \quad (1)$$

where tpm: the number of transactions executed per minute.
And Latency: (time between the execution of the malicious transaction and its detection).

4.3 Experiment 1: Impact of MTDMs on Database Performance

Understanding the impact that the implemented MTDM and the auditing mechanisms have on the database performance is one of the most important features. 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 [13].

Three configurations have been considered in the evaluation of the impact on the database performance: First, Baseline (Oracle 10g fully tuned for performance and without using the audit mechanism); Second, Audit (Oracle 10g using the audit mechanism but without malicious transactions detection); and Finally, The MTDM (Oracle 10g using audit with the malicious transactions detection mechanism).

As the impact in the performance caused by the audit and the implemented MTDM may depend on the transactions submitted to the database, we have decided to measure the performance under the considered four authorized transactions. Fig.4. presents the results for the baseline configuration. 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).

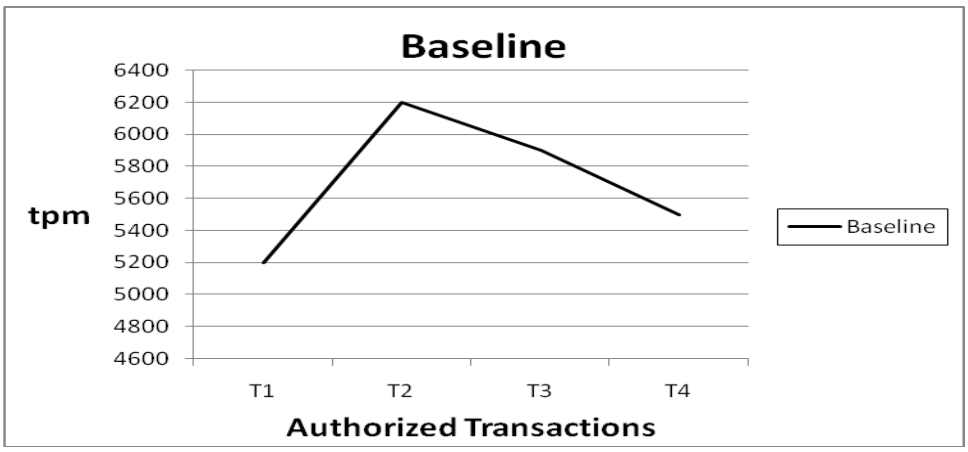

Fig.4. Baseline configuration Performance

Results in Fig.4, show that for the baseline configuration, the number of transactions executed per minute (tpm) is related to the number of commands in each transaction. For example, transactions T2 and T3 achieve an execution rate of about 6200 tpm and 5900 tpm respectively, as transaction T2 contains two commands and transaction T3 contains three commands. While transactions T1 and T4 achieve an execution rate of about 5200 tpm and 5500 tpm respectively, as transaction T1 comprises of eight commands and transaction T3 comprises of four commands. Fig.5. presents the results for the audit configuration.

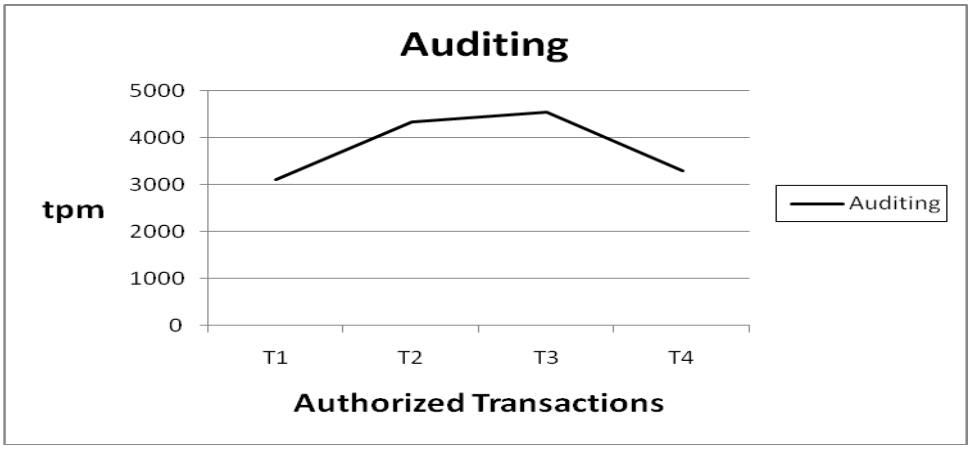

Fig.5. Performance for auditing configuration

Results in Fig.5 show that the impact in the database performance caused by the auditing mechanism is highly related to how much you audit. As shown, the number of transactions executed per minute (tpm) for transactions T1 and T4 are about 3100 and 3200 respectively, as the number of tables that you audit in transactions T1 and T4 are 4 tables. While the number of transactions executed per minute (tpm) for transactions T2 and T3 are about 4200 and 4300 respectively, as the number of tables that you audit in transactions T2 is 1 table and T3 is 2 tables. Figure 6 presents the results for the three configurations considered.

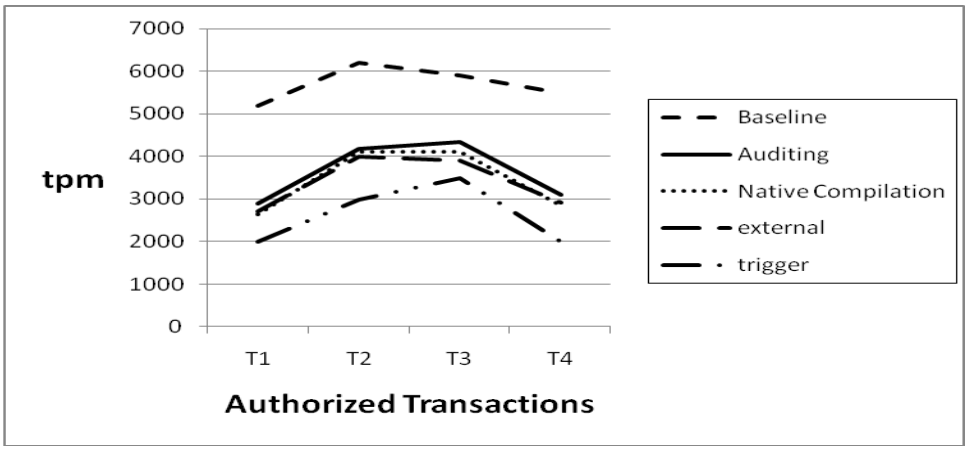

Fig.6. Performance for the three configurations

From the analysis of the results presented in Fig.6, we can derive the performance overhead introduced by the standard auditing mechanism (without malicious transactions detection) and by the three mechanisms. Table 1 shows the resultant overhead.

Table1. Performance overhead

Configuration	Impact %			
	T1	T2	T3	T4
Auditing	40%	33%	35%	40%
Native MTDM	45%	36%	37%	45%
External MTDM	45%	35%	38%	44%
Trigger MTDM	55%	41%	42%	54%

As shown in the previous table, the average performance overhead caused by the activation of the auditing mechanism is fairly high (about 37%), about 40% for the external and the native mechanisms, and about 48% for database trigger mechanism. This explains the expected better performance of the external and native mechanisms over the database trigger mechanism.

Detailed investigations could lead to the following remarks:

- The performance penalty added by the native and external mechanisms is quite small (less than 5% independently on the transaction submitted with respect to the allowable authorized transactions). In other words, the performance penalty is due to the database auditing mechanism.
- The impact on performance is related to how much you audit, so try to decrease auditing (as long as it does not have an adverse impact to database security or your ability to pass an audit). This will also make your reviewing process less tedious, and will require less disk space, etc.
- If your requirements or implementation changes and you change your audit policy in a substantial way, you must go through an entire change management process and comprehensive testing to avoid surprises in production.
- If you want to avoid this overhead (both on the server and more importantly on the rollout of changes to policies) or if you have extreme audit requirements including auditing of DML or SELECT. You either have to accept the performance impact or consider an auditing solution that is not based on the database doing more I/O.
- Finally, remember that the performance impact is not only in the generation of the audit trail itself. The audit trail needs to be moved elsewhere because it can't stay in the database or on the OS. It should be moved fairly quickly for better separation of duties. This means that you also need a process that keeps reading these records, copying them elsewhere, and deleting them. This will add to the impact on performance [13].

4.4 Experiment 2: Latency

Different behaviors concerning database performance and functionality are to be expected for the different mechanisms considered. Fig.8. shows the results obtained for latency time of the three implemented mechanisms. 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 that access and modify the telephone database tables. An important aspect is that, the number of commands in each transaction ranges from 1 to 8.

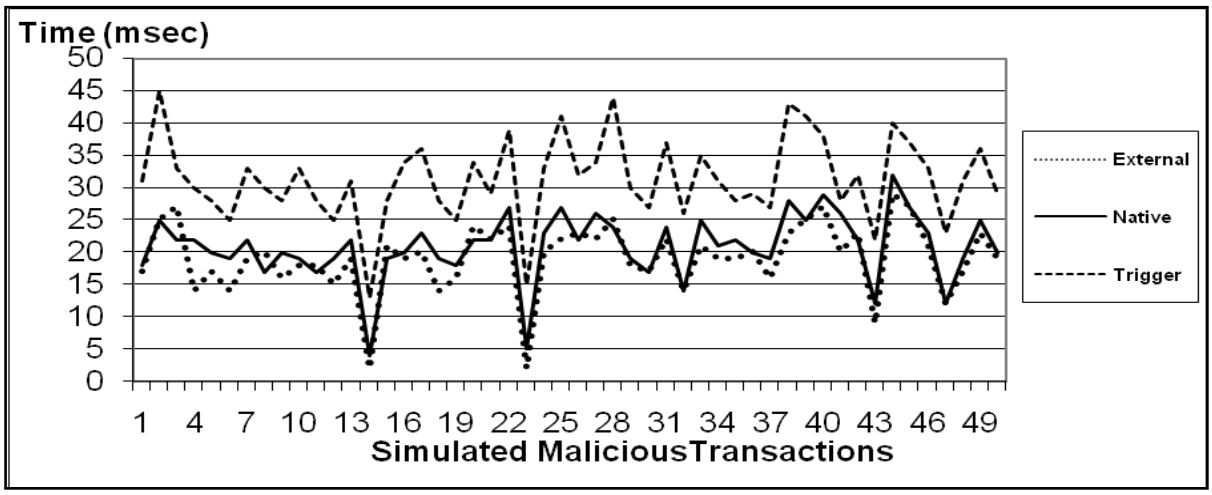

Fig.8. The three mechanisms Latency

From Fig.8, 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 to 29 msec for the external mechanism, between 4 msec to 32 msec for the native mechanism, and between 13 msec to 45 msec for the database trigger mechanism.

As expected, the external and native mechanisms outperform the database trigger mechanism in term of database performance. The average latency for the native mechanism is about 20 msec, about 19 msec for the external mechanism, and about 31 msec for the database trigger mechanism.

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.

5. Conclusion and Future Work

This work presents implementation of three mechanisms for detection of malicious transactions in the Oracle 10g DBMS, and investigates the performance of the three mechanisms using a telephone database. The experimental results showed that the average performance overhead caused by the activation of the external and the native mechanisms is about 40%, and about 48% for the database trigger mechanism, The average latency for the native mechanism is about 20 msec, about 19 msec for the external mechanism, and about 31 msec for the database trigger mechanism. As a result, the external and the native mechanisms outperform the database trigger mechanism in term of database performance (impact and latency).

As a future work, we are planning to propose and evaluate some sort of hyper design that mix between the mechanisms to achieve remarkable optimization between security overhead and performance degradation. The mathematical model might be reviewed for achieving such optimization. Moreover, we will be interested on the external interferences aimed to cause undue delays in accessing or using data, or even denial of service.

6. References

[1] Bertino, E., Sandhu, R., "Database Security Concepts, Approaches, and Challenges" ", *IEEE Transactions on Dependable and Secure Computing*," April 4, 2005, pp. 2-19.

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

[3] Mattsson, U., "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.

[4] Luenam, P., Liu, P., "The Design of an Adaptive Intrusion Tolerant Database System" , *International Conference on Foundations of Intrusion Tolerant Systems,*" 2003, pp. 14-21.

[5] Adams, D., Cowan, M., and Moran, R., Oracle® Database Application Developer's Guide Fundamentals, November 2005, pp. 181-362.

[6] Nagavalli, P., Nathan, P., Introduction to Oracle 9i:PL/SQL, Vol. 2, pp. 210-211, 2001.

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

[8] Michele, C., Oracle® Database Concepts 10g Release 1 (10.1), Redwood, 2003.

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

[10] Shalish, W.K., Salama, G.I., Hesham, A.M., ", Performance evaluation of the detection mechanisms for malicious transactions in DBMS," ", *in 7th International Conference on Electrical Engineering ICEENG 2010,*" Military Technical College, Cairo, Egypt, 25-27 May 2010.

[11] Ayushi., Sharma, A., and Bansal, R., "Detection of Malicious Transactions in DBMS" ,*International Journal of Information Technology and Knowledge Management,*" July-December 2010, Vol. 2, No. 2, pp. 675-677

[12] Deborah, R., Oracle® PL/SQL Programming, 3rd. ed., Vol. 1, united states of America, 2000, pp. 910-915.

[13] Natan, B.N., How to Secure and Audit Oracle 10g and 11g , New work, 2009, pp. 228-229.