Integrated Intrusion Detection in Databases

José Fonseca, Marco Vieira, Henrique Madeira
CISUC, Department of Informatics Engineering
University of Coimbra – Portugal
josefonseca@ipg.pt, mvieira@.dei.uc.pt, henrique@dei.uc.pt

Abstract. Database management systems (DBMS), which are the ultimate layer in preventing malicious data access or corruption, implement several security mechanisms to protect data. However these mechanisms cannot always stop malicious users from accessing the data by exploiting system vulnerabilities. In fact, when a malicious user accesses the database there is no effective way to detect and stop the attack in due time. This practical experience report presents a tool that implements concurrent intrusion detection in DBMS. This tool analyses the transactions the users execute and compares them with the profile of the authorized transactions that were previously learned in order to detect potential deviations. The tool was evaluated using the transactions from a standard database benchmark (TPC-W) and a real database application. Results show that the proposed intrusion detection tool can effectively detect SQL-based attacks with no false positives and no overhead to the server.

Keywords: Databases, security, intrusion detection.

1 Introduction

Traditional database security mechanisms offer basic security features such as authentication, authorization, access control, data encryption, and auditing. However, these mechanisms do not assure protection against the exploitation of vulnerabilities in database management systems (DBMS) and are very limited in defending data attacks from the inside of the organization. In fact, as typical database security mechanisms are not primarily designed to detect intrusions (they are designed to avoid the intruder's access to the data), there are many cases where the execution of malicious sequences of SQL[1] commands (transactions) cannot be detected or avoided.

The general lack of intrusion detection features in typical DBMS is an important limitation when it is necessary to assure a strong data security policy. In fact, intrusion detection in DBMS has not been studied much, in a clear contrast to what has happened in operating systems and networking fields, where many intrusion detection approaches have been proposed. However, malicious actions in a database application may not be seen as malicious by existing intrusion detection mechanisms at the network or the operating system levels. Furthermore, inside attacks are particularly difficult to detect and isolate, as the attacks are carried out by legitimate users that may have access rights to data and system resources. In addition, daily routine and long established habits tend to relax many security procedures. Even

[1] SQL stands for Structured Query Language, the relational language used by relational DBMS [1]

simple things such as choosing strong passwords and periodically purging unused database accounts are often neglected in many organizations [2]. This way, a practical tool for intrusion detection in DBMS that detects malicious behavior from applications and users will provide an extra layer of security that cannot be assured by classic security mechanisms.

According to a FBI Computer Crime and Security Survey [3] in 2005, approximately 45% of the inquired entities had reported unauthorized access to information estimating a loss of $31.233.100 and a loss of $30.933.000 due to theft of proprietary information. Up to 56% reported unauthorized use of computer systems and 13% did not know if they had been attacked. Furthermore, 95% of the inquired entities reported more than 10 web site incidents. These figures show the relevance of an intrusion detection mechanism at DBMS level.

Masquerade attacks where people hide their identity by impersonating other people on the computer are one of the most frequent forms of security attacks [4], including in the database domain. Another common database attack is SQL injection in web applications [5], where unchecked input is sent to a back-end database for execution. The attacker can perform this by simply changing the SQL query sent to the server, accessing sensitive data.

The intrusion detection tool presented in this practical experience report adds concurrent intrusion detection to DBMS. This way, data security attacks can be detected and stopped immediately while the mechanism may call the attention of the database administrator (DBA) by sending an email or an SMS message). This tool, named IIDD - Integrated Intrusion Detection in Databases, works in two modes: transactions learning and intrusion detection. During transactions learning, the IIDD extracts the information it needs directly from the network packets sent from client applications to the database server using a network sniffer. The result is the directed graph representing the sequence of SQL commands that composes the authorized transactions. The learned graph is used later on by the concurrent intrusion detection engine. When an intrusion is detected an alarm is raised and, depending on the tool configuration, the database connection between the intruder and the server may be killed. In this case the connection is abruptly broken, and the database automatically performs a rollback of the malicious transaction. An important aspect is that this tool can be easily used in any commercial-off-the-shelf (COTS) DBMS.

The structure of the paper is as follows. Section 2 presents our approach to intrusion detection in DBMS. Section 3 describes the proposed tool. Section 4 presents two examples of utilization of the intrusion detection tool and Section 5 concludes the paper.

2 Intrusion detection approach

The main goal of security in DBMS is to protect the system and the data from intrusion and unauthorized accesses, 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. 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 (bugs) in the

system implementation. Vulnerability prevention consists of guarantying that the software used has the minimum vulnerabilities possible and this can be achieved by using adequate DBMS software. On the other hand, because the effectiveness of security mechanisms depends on their correct configuration and use, the DBA must correctly configure them by following administration best practices. Vulnerability removal consists in reducing the vulnerabilities found in the system. The DBA must pay attention to the new security patches released by software vendors and install those patches as soon as possible. Furthermore, any configuration problems detected in the security mechanisms must be immediately corrected.

Security attacks are an external factor that mainly depends on the intentionality and capability of humans to maliciously break into the system taking advantage of potential vulnerabilities. The prevention against security attacks includes all the measures needed to minimize (or eliminate) the potential attacks against the system. On the other hand, attack removal is related to the adoption of measures to stop attacks that have occurred before.

General methods for intrusion detection in computer systems are based either on pattern recognition or on anomaly detection. Pattern recognition is the search for known attack signatures in the commands executed. Anomaly detection is the search for deviations from an historical profile of good commands.

Schonlau et al [4] evaluated several anomaly detection approaches and concluded that methods based on the idea that commands not previously seen in the training data may indicate an intrusion attempted, are among the most powerful approaches for intrusion detection. Our intrusion detection approach uses this idea, extending it to a set of SQL commands. However, unlike intrusion detection approaches used in distributed systems that usually rely on sets of predefined commands (normally a small number) or assume the commands are unrelated, in our approach, the SQL commands and their order in each database transaction are relevant.

In spite of all the classical security mechanism developed in the database area, current DBMS are not well prepared for high-assurance privacy and confidentiality [6]. A very important component for the new generation of security aware DBMS is an intrusion detection mechanism [7]. Recent works have addressed concurrent intrusion detection (and attack isolation) in DBMS, and this issue is clearly getting more and more attention.

DEMIDS [8] is a misuse detection system tailored to relational database systems. It uses audit logs to derive user profiles describing typical behavior of users in the DBMS. Chung introduces the notion of distance measure and frequent item sets to capture the working scopes of users using a data mining algorithm. In [9] a real-time intrusion detection mechanism based on the profile of user roles is proposed. An intrusion attack and isolation mechanism was proposed in [10]. This mechanism uses triggers and transaction profiles to keep track of the items read and written by transactions and isolates attacks by rewriting user SQL statements. The use of data dependency relationships and Petri-Nets to model normal data update patterns was proposed in [11] to detect malicious database transactions. DIDAFIT [12] works by matching SQL statements against a known set of valid transactions fingerprints. The algorithm consists in representing SQL as regular expressions using heuristics to assure a low level of false positives. Vieira and Madeira [13] focused on the detection of malicious DBMS transactions by using database audit logs to obtain the sequences of SQL commands executed by users, with the assumption that the transaction profiles was known in advance, and provided manually to the detection mechanism.

Although intrusion detection has already been addressed in the works introduced above, intrusion detection at DBMS level continues to be an open issue. The purpose of the present work is to provide a generic tool that can be used in any database application. This way, the proposed intrusion detection tool is available at [14] for download and public utilization.

2.1 Database transactions profiles

In a typical database environment transactions are programmed in the client database applications, which means that the set of transactions remains stable, as long as the application is not changed. For example, in a banking database application users can only perform the operations available at the application interface (e.g., withdraw money, balance check account, etc). No other operation is available for the end-users (e.g., end-users cannot execute ad-hoc SQL commands). This way, it is possible to use transaction profiles for intrusion detection with a reduced risk of false alarms.

Our intrusion detection tool uses the profile of the transactions implemented by the database applications (authorized transactions) to identify user attempts to execute unauthorized SQL commands. A database transaction is represented as a directed graph describing the different execution paths (sequences of selects, inserts, updates, deletes, and stored procedure invocations) from the beginning of the transaction to the commit or rollback commands. The nodes in the graph represent commands and the arcs represent the valid execution sequences. Depending on the data being processed several execution paths may exist for the same transaction and an execution path may include cycles representing the repetitive execution of sets of commands (a typical example of cycles in a transaction is the insertion of a variable number of lines in a customer's order). The transaction ends with a commit or rollback command. The directed graph representing the profile of valid transactions is used to detect unauthorized transactions, which are seen as invalid sequences of SQL commands. This is done by concurrently analyzing the transactions the users execute and comparing them to the profile of the authorized transactions that were previously learned. To learn the profiles and to detect the malicious transactions the following information is required for each command executed:

- **User name**: name of the user who executes the command;
- **Session ID**: identification of the session established when the client application connects to the database server;
- **Action executed**: text of the SQL command. This includes the identification of the end of the transaction (that is the start of a new transaction), which is forced by a commit or a rollback command;
- **Time stamp of the action**: time stamp of the execution of the SQL command.

An important aspect is that the nodes in the graph do not represent concrete commands since these may differ slightly in different executions. For example, consider the following SQL command to select the data from a given employee: "SELECT * from EMP where job like 'CLERK' and SAL >1000". The job and the salary in the select criteria (*job like ?* and *sal > ?*) depend on the targeted data. This way, instead of considering concrete commands we have to represent those commands in a generic way. The approach consists of parsing the SQL commands

and removing all the parts that are not generic (e.g., numbers and strings between quotes are removed). Also, the characters of the command are changed to lowercase.

The proposed intrusion detection technique does not apply to applications that support the execution of ad-hoc queries, as in this case there are no predefined transactions. However, ad-hoc queries normally target decision support system and are not executed in typical database applications, because this type of queries would ruin the performance of the system. The decision support databases are known as data warehouses [15] and are physically separated from typical databases, because the type of queries executed and the data structures used to store data are completely different from typical databases. End-users that process ad-hoc queries represent a small group (managers and decision making personnel) that use a specific type of databases (data warehouses). Data warehouses are periodically loaded with new data that represents the activity of the business since the last load (normally the periodical loads are done on a daily basis). This is part of data warehouses life-cycle and follows a predefined set of rules that are implemented in the loading applications. Our intrusion detection technique is also applicable in this case because it provides the data warehouse server with the capability to detect malicious actions that try to modify the data.

2.2 Learning the authorized profiles

Transaction learning consists of identifying the authorized transactions and representing those transactions as a directed graph specifying the sequences of valid commands. The goal is to automatically learn the transaction profiles obtained through the reading of the network packets sent by the client applications to the database server and store them as a directed graph to be used in the detection phase. Obviously, the learning process must be executed in controlled conditions that should cover the different database application functionalities and, at the same time, must be free of intrusion attempts (which would potentially lead to the identification of malicious transactions as authorized ones). It is worth noting that all the database transactions must be executed during the learning phase, which should be achieved in a controlled environment virtually free of intrusion attempts (i.e., without the database fully open to all the users). There are three ways to obtain the transactions profiles:

– **Manual profiling** can be easily performed when the DBA knows the execution profile of the client application and the number and size of the transactions is not too high. The DBA can create manually the graphs describing the authorized transactions.
– **Concurrently** during the normal utilization of the application. In this case special attention must be taken in order to guarantee that the application is free of attacks during the learning period. This procedure can be shortened by manually executing some of the functions of the application.
– **Running application tests**. Database applications are often tested using interface testing tools that generate exhaustive tests that exercise all the application functionalities. In most cases these tests are specified by highly-trained testers, but can also be generated automatically [16, 17].

When an end-user (client) connects to the database a session is established. The session information travels across the network and the relevant information can be captured by a well positioned sniffer application. After establishing the session the

user can start executing the client application operations. During execution the application accesses the database through the network sending SQL commands and receiving results. The sniffer reads all that flux of information and retains just what it needs. The capture of information will last until the DBA is confident that all the transactions implemented by the application have been executed at least one time. The sniffer can read the network information that is both encrypted or in clear text. To be able to parse encrypted information, the intrusion detection tool must have access to the decryption function and key. An alternative is to place the intrusion detection tool as a proxy, able to perform secure communication with the database client applications.

One of the key points in the learning phase, and in the detection phase as well, is the detection of the first command and the last command of a transaction. A transaction begins when the previous ends, thus the problem is the detection of the end of a transaction. A transactions is ended explicitly (by a commit or rollback command) or implicitly (by a Data Definition Language (DDL) statement [1]) by the application's code. Fig. 1 shows examples of the graphs generated during transactions learning.

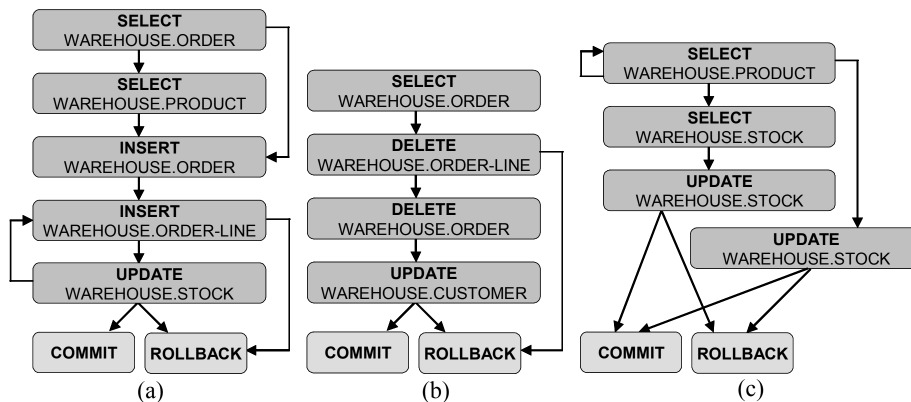

Fig. 1. Examples of typical profiles of database transactions.

2.3 Detecting intrusions

After concluding the learning phase the IIDD tool is able to compare the transactions executed by the users with the authorized profiles described in the transaction graphs. In this phase every command executed must match a profile. When the first command of the transaction is executed the tool searches for all the profiles starting with that same command, which are marked as candidate profiles for the current transaction. The next command executed is then compared with the second command of these candidate profiles. Only those who match remain candidate profiles. This process is executed over and over until the transaction reaches its end or there are no more candidate profiles for that transaction. In this latter case the transaction is identified as malicious. In practice, to detect malicious transactions the IIDD tool implements the following generic algorithm over the transactions graph:

```
while (1) do {
  for each new command executed do {
    if user does not have any active transaction then
      /* command is the 1st command in a new transaction */
      obtain list of authorized trans. starting with
      this command;
    else {
      for each valid (authorized) trans. for the user do {
        if the current command represents a valid successor node
        in the transaction graph then
          the command is valid;
        else
          mark the current transaction as a non valid trans.;
      }
      if there are transactions marked as non valid then
        a malicious transaction has been detected;
    }
  }
}
```

When a malicious transaction is detected one or more of the following actions may be executed, depending on the DBA choice:

- Notify the DBA about the intrusion. The database intrusion detection mechanism is able to provide the DBA with relevant information such as the user name, the time stamp, the database objects damaged, etc. It is also possible to send a message (email or SMS) to the DBA to call his immediate attention.
- Immediately disconnect the user session in which the malicious transaction was attempted and prevent it from logging in again.
- Activate a damage confinement and repair mechanism. When available, damage confinement and repair mechanisms are able to confine the damage and recover the database to a consistent state previous to the execution of the malicious transaction. Another possibility is to isolate that transaction from other user transactions (e.g., by creating a virtual database where the malicious transactions are executed to prevent spreading wrong or malicious data to the database [10]).

3 The intrusion detection tool

Fig. 2 presents the typical environment needed to run our intrusion detection tool. The IIDD tool is installed and runs in the Sniffer Computer. The Database Server network cable must connect to a LAN switch port. The Sniffer Computer must be connected to the span port mirroring of the switch. Switches usually prevent promiscuous sniffing, however, most modern switches support span port mirroring, which replicates the data from all ports onto a single port allowing the sniffer to capture the network traffic of the whole LAN.

The IIDD tool is a two tier application with a back end module and a front end interface (Fig. 3). All the heavy processing is done in the back end which is responsible for sniffing the network searching for packets sent to the database, learn the profiles and detect the intrusions. It is named DBSniffer and was written in C++ to be able to access the network using raw sockets and processing them at the highest speed. This program sends messages through the standard output device and creates several files for future analysis. It is organized into three modules: sniffer, learner and

detector. The IIDD tool can be run in Windows and Linux operating systems (OS) and can be used in any DBMS as the implementation is generic. Both the learning and the detection modules use a common function that is responsible for the detection of network packets.

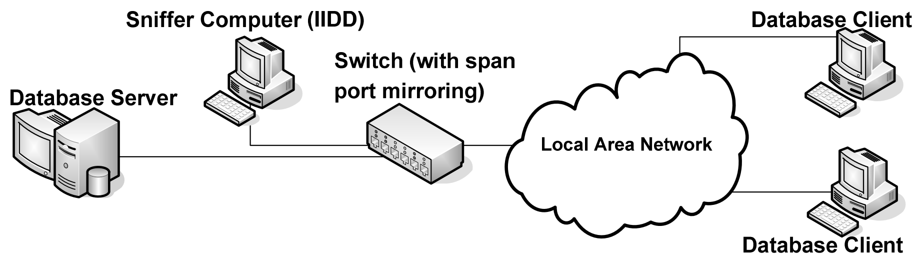

Fig. 2. Architecture setup.

The sniffer is the only DBMS specific component and it is responsible for capturing network packets. Because the tool is based on autonomous components that provide well defined interfaces, it is very easy to implement that function for several DBMS and include it in the tool. For this practical experience report we have chosen Oracle 10G R2 [18] since it is one of the most representative DBMS on the market.

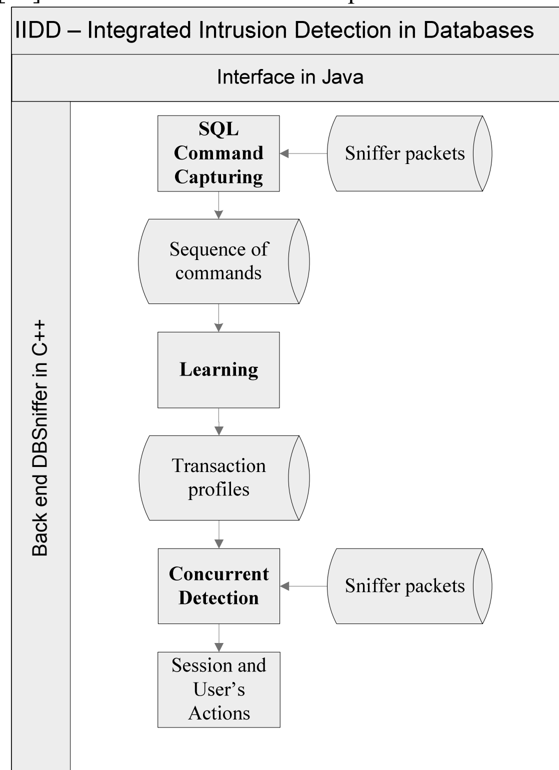

Fig. 3. IIDD tool.

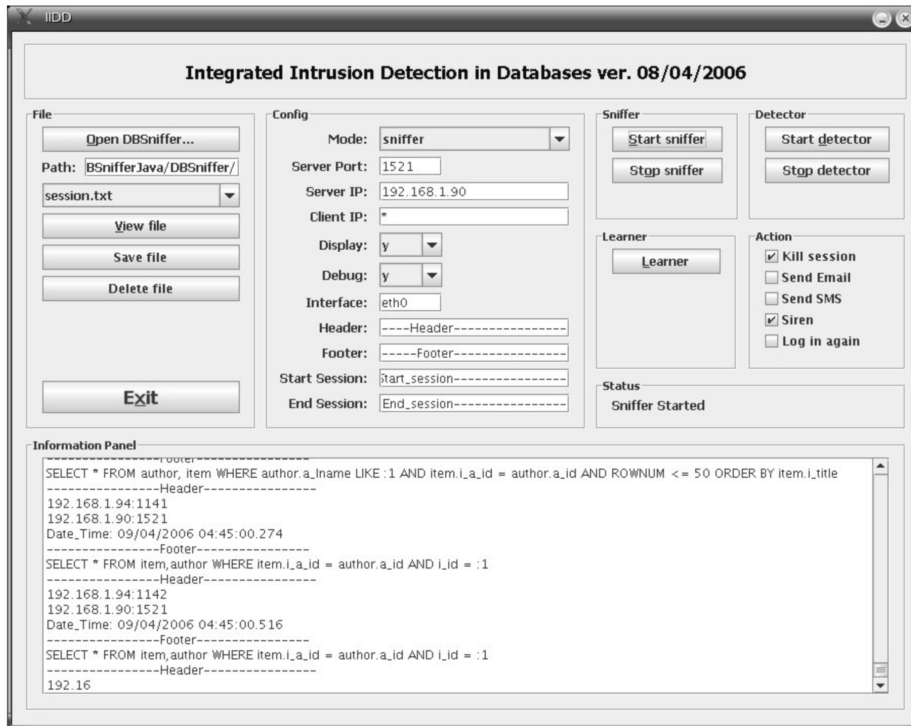

Fig. 4. Interface of the Integrated Intrusion Detection in Databases application.

The sniffer, learner, and detector modules are executed when they are called by the front end application. The front end is a graphical interface, programmed in Java, whose function is to configure and launch the back end and to show the results. The interface has eight groups with different functions: File, Config, Sniffer, Learner, Detector, Action, Status and Information Panel. A screen shot of the prototype's interface is shown in Fig. 4.

The **Sniffer group** starts and stops the sniffer. The sniffer uses raw sockets and places the network adapter in the promiscuous mode. In this mode the network adapter is able to intercept and read all the packets in the network (recall that in non-promiscuous mode the network adapter reads only the packets aimed to it). The output information is copied to the Information Panel. The sniffer module retains only those packets related to the client database communication and saves that information in two files: one with session information (session.txt) and the other with command data (auditory.txt). A debug file may also be created containing all the packet information captured, before any processing is done to the data.

The **Learner group** is used to activate the transaction learning mode. Transaction learning includes two steps: parsing and learning. The former uses the auditory.txt file (generated by the sniffer component) and is responsible for cleaning the commands executed by the database users, removing variable numbers, strings, extra spaces and normalizing the case. After that it generates the file aud.txt. Using this file and the session.txt file the learner algorithm can now be executed. In this phase a file is created with all the transaction profiles (profile.txt). The output information is copied to the Information Panel. This ends the learning phase of our mechanism.

The **Detector group** is used to start and stop the online intrusion detector. For the detection the network adapter is again placed in promiscuous mode in order to sniff all the network packets. The packets are filtered to detect malicious commands compared to the transaction profiles previously learned. It also detects deviations from the order of execution of commands inside the transaction. Those suspicious situations raise warnings immediately, which are saved in a debugging file (detect_debug.txt). The output information is copied to the Information Panel.

The **Action group** allows the configuration of the actions to be executed when a malicious transaction is detected or a transaction is found in a misplaced order. The session maybe killed by sending TCP/IP resets. The connection is abruptly broken, and the database performs a rollback of the malicious transaction in this situation. The DBA may be warned by email, SMS or by a siren sound.

4 Tool utilization examples

The well-known TPC-W transactional web benchmark [19] has been used to exemplify the tool utilization. This benchmark provides us with a controlled database environment quite adequate for the evaluation of the learning and detection algorithms. It simulates the activities of an e-commerce business oriented transactional web server. A real database application is also used in the experiments, the SCE (Serviço Central de Esterilização – Central Service of Sterilization). It is an administrative application currently in use in the Hospitals of the University of Coimbra (HUC: http://www.huc.min-saude.pt/) used to manage the whole process of the sterilized material to and from all services in the HUC. This workflow comprises the reception of the material, the selection and the sterilization of the material within a central with steam autoclaves and ethylene oxide, various modes of drying, packaging, sealing, request and delivery.

To obtain the latency and the coverage information we have built a SQL command line tool that records the time stamp of the command request and the time stamp of the server response. It is also capable of some configuration to assist the coverage experiments. This SQL command tool obtains a list of the commands (executed by the client application being monitored) using the text files created by the learner module. With this list we can inject real SQL commands used by the client application into the DBMS while the intrusion detection is active. This command injection may contain the exact command text, or a slightly altered command, injected using a random order. It can also introduce small random changes in the command to test the efficiency of the detection mechanism. The statistical distribution of utilization of each command is equal to the distribution of the command when executed by the application being monitored.

The TPC-W work-load was executed during one hour to learn the transaction profiles (Fig. 5). As a result 8971 transactions containing 18834 commands were executed and 32 transaction profiles were learned. As expected, the learning curve rises abruptly in the first transactions executed and then its trend is to stabilize over time. After the learning phase, TPC-W was executed for three hours to check if the learning was exhaustive. No transactions were detected as intrusion, which indicates that all transactions were correctly learned.

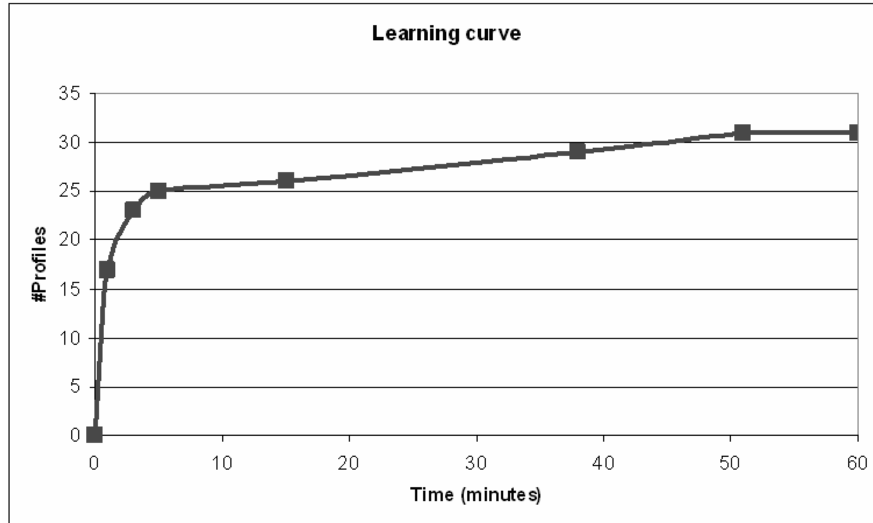

Fig. 5. TPC-W learning curve.

As the sniffer is located in a different computer it has no impact on the server performance, thus this mechanism has no performance overhead. Furthermore, the mechanism does not introduce extra packets in the network, causing no negative effect in the network bandwidth.

With our SQL command tool we manually injected 50 malicious commands while the TPC-W was running to produce the load. They were very simple commands to be quickly executed by the server. All these 50 commands were detected as intrusions. The largest latency time obtained was 10 milliseconds and the average latency time was 1.6 milliseconds. However these times may vary with the network setup and usage. All detections were performed before the server had replied to the client, thus before the execution of the next command.

We also performed two additional experiments with extraneous SQL commands injection: in the first we injected the correct commands, but in a random order; and in the second we made a change in one character randomly placed in each command. The distribution of utilization of each command was the real distribution of the command when the TPC-W executes it. For the first experiment 1000 transactions were executed with a total of 2061 commands. For the second experiment 1000 transactions were also executed with a total of 2066 commands. All those transactions were considered malicious resulting in 100% coverage.

Besides the TPC-W benchmark, we performed some experiments with a real database application: the Central Service of Sterilization database mentioned before. The main goal was to evaluate the learning algorithm in a real database scenario, helping us to assess the learning transaction curve and to estimate false positives caused by incomplete learning. We used the information of one working day of real utilization of the database of the SCE, having 8750 commands from 609 sessions and accesses 17 tables. This log was applied to the Learning module and 33 transaction profiles where learned.

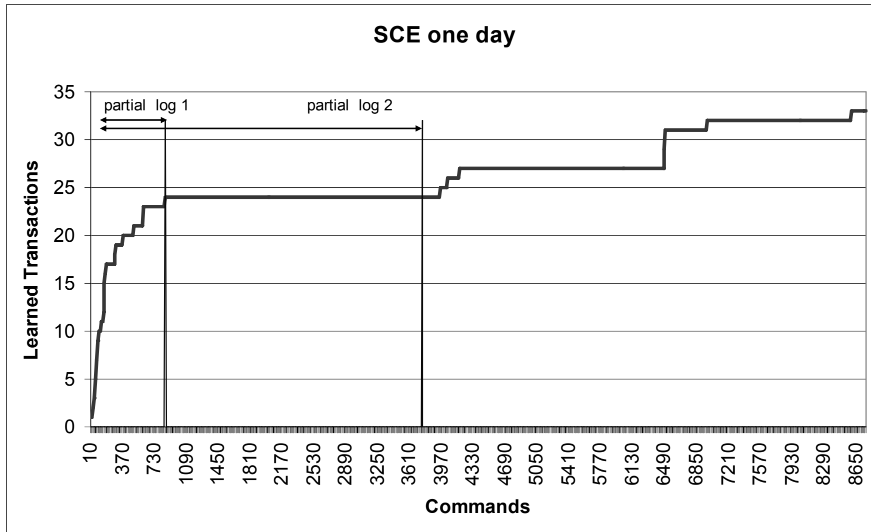

Fig. 6. SCE transaction learning during one day

Fig. 6 shows the learning transaction curve. As we can see, most of the transactions (24 out of 33) were learned very quickly, during the first 858 commands (partial log 1). It is also quite evident that two new groups of database functionalities (and corresponding transactions) were executed around the command number 4000 and command number 6500, corresponding to the two steps in the learning curve. If we had learned only from the partial log 1 or even the partial log 2 it would be necessary to make a conditional detection in those periods of time, as all transactions not learned in log 1 and log 2 would be detected as false positives. In that case, the database administrator would have to add the new transactions to the profile collection, in order to avoid further detection of these transactions as false positives. In a typical database application there are moments in time where some specific procedures are executed (e.g., at the end of the day, at the end of the week, etc) and the time window used to learn the profiles must contain those moments.

An important aspect is that a one day learning period is quite small when we are considering applications that are used during many years (as is the case of SCE). In fact, a larger period is required as some operations are executed in specific moments in time. For the SCE, further experiments showed that a learning period of about a week is required to learn almost all transactions.

5 Conclusions

In this paper we present a new tool for the detection of malicious transactions in DBMS. This database intrusion detection mechanism uses a graph that represents the profile of valid transactions to detect unauthorized transactions and consists of three different components: SQL command capture, transaction learning and concurrent intrusion detection. This tool is generic as it can be used in any typical DBMS, including stat-of-the-art commercial DBMS. The setup used to exemplify the tool

utilization consists of a typical database environment using an Oracle server running the workload from the TPC-W benchmark and a real database application. The results show that the proposed tool is quite effective and can be easily used by the DBA.

References

1. C. J. Date and Hugh Darwen, "The SQL Standard", 3rd Edition (Addison-Wesley Publishing Company, 1993), 414 pages; paperbound; ISBN 0-201-55822-X.
2. Andrew Conry-Murray, "The Threat From Within", http://www.itarchitect.com/shared/article/showArticle.jhtml?articleId=166400792 (2005)
3. Lawrence A. Gordon, Martin P. Loeb, William Lucyshyn and Robert Richardson, Computer Security Institute. Computer crime and security survey (2005)
4. M. Schonlau, W. DuMouchel, W.-H. Ju, A. F. Karr, M. Theus, and Y. Vardi, "Computer intrusion: Detecting masquerades", Statistical Science, 16(1):58–74 (2001)
5. Moran Surf and Amichai Shulman, "How safe is it out there? Zeroing in on the vulnerabilities of application security", Imperva Application Defense Center Paper (2004)
6. A. Anton, E.Bertino, N.Li, and T.Yu, "A roadmap for comprehensive online privacy policies", In CERIAS Technical Report, 2004-47 (2004)
7. R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Hippocratic databases", In Proceedings of the 28th international conference on Very Large Data Bases, Morgan-Kaufmann (2002)
8. Christina Yip Chung, Michael Gertz, Karl Levitt, "DEMIDS: A Misuse Detection System for Database Systems", 3rd IFIP TC-11 WG11.5 Working Conference on Integrity and Internal Control in Information Systems, Kluwer Academic Publishers, 159 – 178 (1999)
9. Elisa Bertino, Ashish Kamra, Evimaria Terzi, Athena Vakali, "Intrusion detection in RBAC-administered databases", 21st Annual Computer Security Applications Conference (2005)
10. Peng Liu, "DAIS: A Real-time Data Attack Isolation System for Commercial Database Applications", In Proc. of the 17th Annual Comp. Security Applications Conf. (2001)
11. Y. Hu, B. Panda, "Identification of malicious transactions in database systems", the International Database Engineering and Applications Symposium (2003)
12. Sin Yeung Lee, Wai Lup Low, and Pei Yuen Wong, "Learning Fingerprints for a Database Intrusion Detection System", 7th European Symp. on Research in Computer Security (2002)
13. M. Vieira, H. Madeira, "Detection of malicious transactions in DBMS", the 11th IEEE Intl Symposium Pacific Rim Dependable Computing, (2005)
14. J. Fonseca, M. Vieira, H. Madeira, "Tool for Integrated Intrusion Detection in Databases", available at: http://gbd.dei.uc.pt/downloads.php (2007)
15. R. Kimball, Ed. J., "The Data Warehouse Lifecycle Toolkit", Wiley & Sons, Inc, (1998)
16. V. Santiago, A. Amaral, N. L. Vijaykumar, M. Mattiello-Francisco, E. Martins, O. Lopes: "A Practical Approach for Automated Test Case Generation using Statecharts", 30th Annual International Computer Software and Applications Conference, 2006, Chicago, (2006)
17. W.T. Tsai, X. Bai, B. Huang, G. Devaraj, R. Paul: "Automatic Test Case Generation for GUI Navigation", in The Thirteenth International Software & Internet Quality Week (2000)
18. Oracle Corporation, "Oracle® Database Concepts 10g Release 1 (10.1)" (2003)
19. Transaction Processing Performance Council, "TPC Benchmark W, (Web Commerce) Specification, Revision 1.8", available at: http://www.tpc.org/tpcw, (2002)