Approaches and Challenges in Database Intrusion Detection

Ricardo Jorge Santos
CISUC – DEI – FCTUC
University of Coimbra
3030-290 Coimbra – Portugal
lionsoftware.ricardo@gmail.com

Jorge Bernardino
CISUC – DEIS – ISEC
Polytechnic Institute of Coimbra
3030-190 Coimbra – Portugal
jorge@isec.pt

Marco Vieira
CISUC – DEI – FCTUC
University of Coimbra
3030-290 Coimbra – Portugal
mvieira@dei.uc.pt

ABSTRACT

Databases often support enterprise business and store its secrets. This means that securing them from data damage and information leakage is critical. In order to deal with intrusions against database systems, Database Intrusion Detection Systems (DIDS) are frequently used. This paper presents a survey on the main database intrusion detection techniques currently available and discusses the issues concerning their application at the database server layer. The identified weak spots show that most DIDS inadequately deal with many characteristics of specific database systems, such as *ad hoc* workloads and alert management issues in data warehousing environments, for example. Based on this analysis, research challenges are presented, and requirements and guidelines for the design of new or improved DIDS are proposed. The main finding is that the development and benchmarking of specifically tailored DIDS for the context in which they operate is a relevant issue, and remains a challenge. We trust this work provides a strong incentive to open the discussion between both the security and database research communities.

1. INTRODUCTION

Databases are of vital importance to nearly all enterprises. They support the business' operational and analytical requirements, and often store its secrets. For example, Data Warehouses (DWs) store extremely sensitive business information, making them a major target for attackers. Therefore, securing their data from damage or leakage is a critical issue. To manage this, enterprises typically implement several layers of protection between users and data, working at the network, host, and database levels.

Most solutions for data protection at the database layer consist of Database Intrusion Detection Systems (DIDS), well-defined data access policies and encryption. Although data access policies and standard encryption algorithms are widely used, and relatively simple to configure in today's DataBase Management Systems (DBMS), choosing which DIDS to use in certain environments is not a trivial task.

This paper describes and analyzes the main techniques for DIDS, and discusses their practical limitations. For example, since most database intrusion detection techniques rely on command-syntax analysis for user profiling (which typically consists of determining usual data access patterns and dependencies), the *ad hoc* nature of significant portions of data warehousing workloads makes distinguishing abnormal from normal user activity an extremely difficult task. Moreover, most DIDS are incapable of preventing or stopping a user action before it finishes its execution, *i.e.*, they lack intrusion response capability. Given the value of data in many business contexts, these are critical issues that might make the currently available DIDS inefficient or even unfeasible solutions in many database systems.

The main contributions of this work are the description of the distinct existing intrusion detection techniques and a discussion on how these DIDS are applicable to each database context, pointing out their weak spots considering the typical user workload and the specific characteristics of each type of environment. Based on those weaknesses, we present the existing research challenges and opportunities, and propose a set of requirements and guidelines to drive the development of new and/or improved DIDS specifically designed for those environments. We argue that this is a pertinent issue and remains a challenge.

Another important finding is that despite the importance of the role played by benchmarks in testing and comparing systems, until this moment no benchmark has been proposed for evaluating the performance of DIDS at the database level.

2. DATABASE INTRUSION DETECTION SYSTEMS

Detecting illicit access and malicious actions are the main goals of Intrusion Detection Systems (IDS). There are mainly two approaches: *misuse detection*, looking for well-known attack patterns; and *anomaly detection*, looking for deviations from typical user behavior. The first approach works efficiently against previously known and expected intrusion actions.

However, it is incapable of acting against intrusions that reveal new forms of attack or malicious user actions that seem "normal". To overcome these issues, anomaly detection techniques have been proposed.

In these systems there is typically a learning or training phase (*i.e.*, previous to intrusion detection), in which database logs and/or command datasets assumed as having "normal" or intrusion-free activity are used in order to build the user behavior profiles [21]. After this learning phase, the intrusion detectors match user actions against those profiles to find significant deviations which are signaled as potential intrusions.

The main requirements that intrusion detection systems need to cope with are:

1) Adequately defining and building profiles that accurately represent "normal" user behavior or workloads, as well as identifying attack signatures;
2) Given those profiles and/or attack signatures, define which behavioral features as well as which techniques and models maximize the performance and accuracy of the intrusion detection processes;
3) Reporting system status to security staff and notifying them about generated alerts;
4) Promote a way of stopping or preventing the attack whenever an intrusion alert is raised (this feature may or not be present in the IDS; if it is the case, literature often refers the IDS as an *Intrusion Detection and Response System*, or *Intrusion Detection and Prevention System*).

From an impersonation perspective, an intruder can be one of the following [30]:

- An *authorized user*, which is someone belonging to the enterprise that has regular access to authorized database interfaces and acts with malicious intent (commonly referred as an *insider threat*);
- A *masqueraded user*, which is someone that obtains the credentials of an authorized user and impersonating that user takes control of an authorized interface (also referred as an *insider threat* when the attacker is someone from within the enterprise but without regular authorized database access, and refers to an *outsider threat* when it comes from someone outside the enterprise that manages to obtain the credentials);
- An *external attacker* (commonly referred to as the *outsider threat*), which is someone from outside the enterprise that is able to bypass database security and gain direct database access using SQL injection or other vulnerability exploiting techniques.

Considering the intruders' intentions, there are mainly three types of attacks mobilized against databases [8]:

- *Attacks aiming at corrupting data (integrity attacks)*. In these types of attack, the intruder seeks access to the database for executing actions that compromise its integrity, such as corrupting or deleting the data in a given database object (*e.g.* such as modifying the contents of a table);
- *Attacks aiming at stealing information (confidentiality attacks)*. In these attacks, the intruder focuses on breaking confidentiality issues, such as stealing business information, rather than damaging data;
- *Attacks aiming at making the database unavailable (availability attacks)*. These attacks aim on making database services unavailable to users, *i.e.*, they are mainly Denial of Service (DoS) attacks (*e.g.* flooding database services and bandwidth with a large number of requests, crashing database server instances, deleting database objects, etc).

In the past, several types of database intrusion detection techniques have been proposed. This section presents a descriptive analysis of selected samples from each different type of approach and/or technique for dealing with all types of attacks, in order to characterize the broad scope of existing solutions against both insider and outsider threats.

2.1 Temporal Analysis

These techniques focus on temporal features such as the time span between user actions and the duration of those actions. The approach in [18] uses a mean and standard deviation model built from time signatures to check for outliers within a predefined range in real-time database systems. This solution considers a transaction as a set of read and/or write actions for each data object which is executed in predefined update time periods.

For example, updating a temporal data object (event) can trigger a rule such that the update time is checked against the expected update time (condition) and rejects the update (action) if the predicate returns false, considering it an intrusion. The training period occurs until a significant mean with 99% confidence level of a normal distribution is obtained for each object/update pair. Database behavior is monitored by sensors at the transaction level, which are assumed to be small in size and have fixed semantics such as write-only operations and well-defined data access patterns. If a transaction tries to update a temporal data object that has already been updated in that period, an alarm is raised.

2.2 Dependency and Relation Analysis

Intrusion detection techniques based on dependency and relation analysis compute dependencies and/or relations among the distinct sets of user actions and/or accessed data to find out which columns, rows, tables, etc. and/or commands are usually issued or processed together.

The DEMIDS system [4] builds user profiles based on their activity by determining frequent itemsets from feature/value pairs and computes distance measures of user activity against the learnt frequent itemsets to detect intrusions, given a threshold. The features are typically based on the syntactical analysis of user commands, where the itemset domains are the sets of attributes issued together.

Another approach using frequent itemset mining is presented in [33]. This approach summarizes each user command into a tuple <Op, F, T, C> where Op is the type of SQL command (insert, select, etc), F is the set of attributes, T is the set of tables, C is the constrained condition set. An algorithm mines user query profiles using these tuples, based on the pattern of the submitted queries at the transaction level. The algorithm adapts the support and confidence of association rule mining by adding query structure and attribute relations to the computation.

The Role-Based Access Control DIDS proposed in [11] improves a previous approach [1] using features named quiplets for summarizing each user command. Considering a generic command `SELECT {Target-List} FROM {Relation-List} WHERE {Qualification}`, a quiplet is defined as (`C, PR, PA, SR, SA`) where `C` is the SQL main command (insert, select, etc.), `PR` is the Projection-Relation information, `PA` is the Projection-Attribute information, `SR` is the Selection-Relation information, and `SA` is the Selection-Attribute information.

The authors define three types of quiplets with different granularities: given a relation (alias table) `R1` with attributes `A1, B1, C1, D1` and a relation `R2` with attributes `A2, B2, C2, D2` and a user command `SELECT R1.A1,R1.C1,R2.B2, R2.D2 FROM R1,R2 WHERE R1.B1=R2.B2`, they generate the coarse c-quiplet (`select,<2>,<4>,<2>,<2>`), medium m-quiplet (`select,<1,1>,<2,2>,<1,1>,<1,1>`) and fine f-quiplet (`select,<1,1>,<[1,0,1,0],[0,1,0,1]>,<1,1>,<[0,1,0,0],[0,1,0,0]>`).

For anomaly detection when the database has role-based users (*i.e.*, it is possible to link each user action to a given role), a Naïve Bayes Classifier (NBC) is used as follows: for all queries in the audit logs, and for each role, the classifier for each type of quiplet is built (training phase); for each submitted query, if any of its classifiers is different from the ones in its roles, the action is considered an intrusion and an alert is generated (testing phase).

If role-based access policies are not implemented in the database, they propose unsupervised anomaly detection. In this case, positional and distance functions are defined for the quiplets and clustering techniques (*k*-centers and *k*-means) map every user to its representative cluster, which is the cluster with the highest number of training records for that user after the clustering phase (training phase). For each new query to test, two approaches can be used: 1) given the determination of its representative cluster, use the NBC as in the Role-Based anomaly detection to perform a similar test; or 2) verify if the new query is a statistical outlier using the MAD (Median of Absolute Deviations) test [24], which if true considers the action as an intrusion and generates an alert.

2.3 Sequence Alignment Analysis

Sequence alignment mainly consists in determining common sequences of events (such as commands, data attributes, accessed values, etc). DIDS using this type of techniques typically learn and identify the repeatable series of events with significant length and eventually break them into smaller-sized subsets to label or classify those sequences and their subsets as normal user behavior. In the detection phase, each sequence of new events is matched against the learnt user sequences and their subsets for measuring how they differ in order to evaluate its probability of being an intrusion.

The solution presented in [15] identifies sequences of accessed attributes, commands and tables for building user profiles. The proposed features are the command types (insert, select, update, etc.), attributes designated as sensitive, all attributes, operations on attributes, and mixes of all features. This work also defines criteria to choose among user-based, role-based or organization-based profiles, given the working context of the database. In the learning phase, it builds sequence models given a threshold for determining the maximum number of differences. In the detection phase, it also uses a threshold for computing the highest number of differences allowed between the tested sequences and those retained in the learning phase, to consider the sequences as normal or abnormal.

2.4 Integrating Dependency with Sequence Alignment Analysis

An approach for finding dependency relationships among transaction-level attributes with high support and confidence rules is proposed in [10]. They assume that whenever an attribute is updated, this action is

linked to a sequence of other events logged in the database (*e.g.* due to an update of a given attribute, other attributes are also read or written). Thus, each update is defined by three sets: the read set, a set of attributes that have been read because of the update; the pre-write set, a set of attributes that have been written before the update and because of it; and the post-write set, a set of attributes that have been written after the update as a consequence of it. Transactions that do not follow any of the mined data dependency rules are marked as malicious.

The work in [28, 29] improves that of [10] by considering attribute sensitivity, *i.e.*, giving a measure of importance to each attribute. It proposes three levels of attribute sensitivity, given its support in the analyzed transactions: high, medium and low. A weighted data mining algorithm is used to mine the dependencies between database attributes and generate rules that reflect that dependency, given the measured sequences of operations (read, write) and the sensitivity of each attribute. Any transaction not following these rules is identified as malicious. The authors also propose an extension to the Entity-Relationship (E-R) model to syntactically capture the sensitivity of the attributes.

A learning algorithm for representing transactions by directed graphs describing execution paths is proposed in [9]. New transaction sets that deviate from the learnt execution paths are seen as unauthorized sequences of SQL commands. The features used to build the execution paths are the command type (select, insert, delete, etc.), target objects (tables) and selected columns, and restriction attributes, all of which are obtained from typical DBMS audit entries [21] storing information on the UserID, SessionID, CommandID, TransactionID, user command, object owner, and a timestamp of its execution.

2.5 Statistical Analysis

Statistical analysis is used in several DIDS for computing user activity statistics. The approach in [27] makes use of statistical functions on reference values obtained from the data in relations (alias tables) and Δ-relations (changes of the values of the monitored objects/attributes for all reference values, per attribute, between two runs of the DIDS) for anomaly detection. An extension is defined as the set of all rows of an insertion/modification of data and a relation refers to a table or view. The reference values include count, minimum, maximum, average, standard deviation, ranges, computed ratios, zero length checking, and bit counting. A misuse detection method is also included, which works by examining database objects (Database, Function, Index, Privilege, Procedure, Rule, Schema, Statistics, Table, Trigger, and View) and all operations on them. This is done by previously defining if each pair <Database object, operation> is dangerous or not.

The work proposed in [19] is based on computing summarized statistics such as counting, maximum, minimum, mean, median, standard deviation and cardinality values of each attribute from the dataset resulting or affected by each user command. These statistics are stored in a vector with fixed dimension named as S-Vector, regardless of how large the command's result dataset may be. When the dataset for obtaining the S-Vector is large, the authors propose sampling the dataset by fetching the first initial k tuples or a subset of randomly picked k tuples to maintain performance and scalability. The set of each user's S-Vectors is then used for applying techniques such as clustering, naïve Bayes, support vector machines or decision trees in order to obtain models that represent the user's normal behavior given those S-Vectors. In the intrusion detection phase, statistical deviation and outlier verification is applied to inspect each user command and classify it as normal or abnormal.

2.6 Information-Theoretic Analysis

Approaches using information-theoretic analysis compute measures such as entropy and information gain for characterizing user profiles and compare them with those of subsequent actions to see how they differ from the original ones.

The work in [17] describes such an information-theoretic solution. Features are composed by a tuple of audit data with n variables for each data object (*e.g.* IP address, message size, etc). Entropy is used as a measure of regularity of audit data (*e.g.* event types such as a list of commands), where each record represents a class; the smaller the entropy, the fewer the number of distinct records (*i.e.*, there is a higher number of redundancies), indicating more regular audit datasets. The fact that many events are repeated (or redundant) in a dataset suggests that they are likely to appear in the future. Anomaly detection models built from datasets with small entropy will likely be simpler and have better detection performance.

Conditional entropy is used to define temporal sequences of audit data. $H(X|Y)$ shows how much uncertainty remains for the remaining audit events in a sequence X after seeing Y. For anomaly detection, it is used as a measure of regularity of sequential dependencies. If the audit trail is a sequence of events of the same type, then the conditional entropy is 0 and the event sequences are deterministic. Conversely, large conditional entropy indicates that the sequences are not as deterministic and hence much harder to model.

Relative conditional entropy between distributions is used to measure regularities (distance) between two audit datasets, where the training dataset is a validated audit dataset and the tested dataset is the one to be inspected. Once again, the best solution is the one with smaller relative conditional entropy. Information gain is introduced to aid the feature selection and construction process to improve the detection performance because of its direct connection with conditional entropy. The higher information gain owned by the feature, the smaller conditional entropy, and hence the better detection performance.

2.7 Command Template Analysis

Command modeling DIDS use a command log to analyze all regular user commands and build summarized templates that generically represent the typical user workloads.

In [16], an algorithm summarizes a set of supposed "legitimate" queries into SQL templates that represent the models of all the queries. Each conditional filtering variables in the WHERE clause of similar commands are considered as parameters. To see if an unbounded variable or a finite list of values should be used for each parameter, a Kolmogorov-Smirnov test is done at a 90% confidence level. The algorithm also tabulates the frequency of each learnt fingerprint, *i.e.*, how often it occurs in the set of SQL statements.

Taking a new fingerprint `F` and a previously defined fingerprint `F′`, `F` is considered legitimate if `F` differs from `F′` only by: 1) any extra conditions in the WHERE clause of `F` that are missing from `F′` are joined with the AND operator; and 2) `F` selects an equal or fewer number of columns than `F′`. This work also proposes a method to deduce missing fingerprints (*i.e.*, ranges of queries similar to the database log queries used in the learning phase), based on mixing the possible combination of conditions in the WHERE clause from the previously acquired fingerprints. In the testing phase, each command significantly differing from the computed fingerprints is considered abnormal.

In [2] the authors propose applying a grammar-based analysis using tree-kernel based machine-learning techniques instead of commonly used vector-based data. This approach uses the parse-tree structure of SQL for correlating commands with applications and to differentiate between benign and malicious ones by inspecting changes in command syntax trees. They derive a distance measure induced by a tree-kernel function to measure the similarity of SQL commands using their parse-trees. Support vector machines are used in the learning phase and clustering is applied for distinguishing benign from malicious commands by outlier detection. This method promotes a context sensitive similarity that enables locating the nearest non-intrusive command for a malicious statement, which helps in root cause analysis.

3. INTRUSION RESPONSE AND PREVENTION

In what concerns intrusion response and prevention, which is the capability of stopping the intrusion action when it occurs or even before it occurs, it can be seen that several solutions enable full intrusion prevention, while others can only partially accomplish this.

In [18], the temporal analysis technique detects any queries that request execution outside a predefined time schedule and may therefore deny their execution and prevent the intrusion action. The sequence analysis technique used in [15] may enable intrusion prevention by avoiding subsequent user actions when it detects a suspicious sequence of actions. However, it needs to wait for a significant amount of actions that make up that sequence, meaning that it will probably only detect the intrusion after some of those actions have finished their execution, which makes it only capable of partial intrusion prevention.

All the solutions based on dependency and relational analysis that were described [1, 4, 11, 33] are fully capable of enabling intrusion prevention, since they may check each individual user command syntax and if they find those commands suspicious their execution can be stopped before their execution occurs. The solutions integrating a mix of dependency and sequence analysis such as [9, 10, 28, 29] are capable of performing only partial intrusion prevention, for the same reasons pointed out in the previous paragraph concerning the solution proposed in [15].

The work in [12] proposes a DIDS with intrusion detection and response mechanisms, improving a previous proposal in [11]. They propose defining database response policies and deal with potential intrusions using policy matching. The authors propose set of SQL-like rules in a syntax as *ON {Event} IF {Condition} THEN {Action} CONFIRM {Confirmation Action} ON SUCCESS {Resolution Action} ON FAILURE {Failure Action}* that will enable security staff to define those policies and determine what sort of actions the DIDS should take against intrusions.

The anomaly attributes used as intrusion detection features are the UserId, his/her role, client application, source IP address, and date/time of each user action, and the database, schema, object type (table, view, etc), the SQL command and its attributes. An administration model is included to manage the response policies and present algorithms for efficiently searching the policies

matching an anomalous user request in the policy database. The possible responses to intrusion actions can be {Do nothing, Log anomaly details, Send notification, Taint or Suspend user action, Abort or Disconnect user, Revoke or Deny user privileges}.

The solutions presented in [19, 27], based on statistical analysis, are mostly incapable of intrusion prevention, as they mostly rely on analyzing the changes in data or execution results *after* they have been processed. This means they can only detect the intrusion *a posteriori* to the attack. However, the approach in [27] can be adapted to check *a priori* statistical data concerning the rows requested to be processed, enabling partial intrusion prevention capabilities. For this same reason, the information-theory analysis approach presented in [17] may also accomplish partial intrusion prevention.

The solutions based on command and template analysis in [2, 16] can fully enable intrusion prevention due to same reason as those previously mentioned for dependency and relational analysis [1, 4, 11, 33].

Besides the previously described specific intrusion detection techniques and approaches that can be used in databases, other research works have been published that can also contribute to this intrusion detection field. For example, although it does not present itself as a DIDS, the work in [20] describes a method for auditing SQL queries to measure their suspiciousness from a privacy and confidentiality perspective that may be useful for intrusion detection purposes. A generic survey on how data mining techniques can be applied to intrusion detection is shown in [23], and an extensive survey on SQL injection is given in [14].

Table 1 summarizes the techniques described in this section, referring each type of technique along with the actions and user action elements that can be analyzed. It also shows if each approach allows implementing intrusion prevention, *i.e.*, if it enables stopping the intrusion action *a priori* to its execution.

4. APPLICATION OF INTRUSION DETECTION IN DATABASES

The applicability of DIDS in database systems depends on the type of environment in which they are supposed to operate. Understanding the characteristics inherent to the typical workloads of each type of environment is critical to determine which type or class of Intrusion Detection (ID) techniques can be more efficient given the nature of those workloads and thus, be considered as more adequate for the specific database system.

4.1 Transactional versus Analytical Database Systems

In an enterprise, the transactional (alias operational) systems typically consist of a set of applications and data sources that enable accomplishing and storing business transactions, and guarantee their operability [13]. Transactional databases are designed to manage the data required in supporting individual business transaction instead of cross-enterprise business analysis. Transactional systems typically consist of many users reading and writing small amounts of data. For example, an ATM bank system can have hundreds or thousands of users accessing their account balances at the same time or withdrawing/transferring a given amount of money. Another characteristic of the system is that it does not require keeping long periods of historical data; it only needs the current balance and latest movement records to be able to adequately support user requests and business transactions.

Table 1. Database intrusion detection techniques and their coverage

Technique	Reference	Elements that can be analyzed				Intrusion Prevention Capability
		Command Syntax	Accessed Columns	Processed Rows	Result Dataset	
Temporal Analysis	Lee, 2000 [18]	X				Yes
Dependency and Relation Analysis	Chung, 1999 [4]	X	X			Yes
	Zhong, 2004 [33]	X	X	X		Yes
	Bertino, 2005 [1]	X	X			Yes
	Kamra, 2008 and 2010 [11, 12]	X	X			Yes
Sequence Alignment Analysis	Kundu, 2010 [15]	X				Partial
Integrated Dependency with Sequence Alignment Analysis	Hu, 2004 [10]	X	X			Partial
	Srivastava, 2006 [28, 29]	X	X			Partial
	Fonseca, 2008 [9]	X	X			Partial
Statistical Analysis	Spalka, 2005 [27]	X	X	X		Partial
	Mathew, 2010 [19]	X	X		X	No
Information-Theoretic Analysis	Lee, 2001 [17]	X				Partial
Command Template Analysis	Lee, 2002 [16]	X	X			Yes
	Bockermann, 2009 [2]	X	X			Yes

In contrast, analytical systems are usually accessed by fewer users that query large amounts of data to obtain business analysis information to aid decision making. Using the same bank ATM system as an example, the difference is that the people from the bank that need to make decisions regarding the business (*i.e.*, managers, administrators, etc.) want to know the average balance for the last six months or a year for the accounts within a certain geographical region, for instance, in order to aid strategic decisions like opening a new branch office or encourage people to increase their investments by offering better interest rates. To execute this kind of query, the system needs to keep historical data of the balances plus it would read millions of records of all clients within a certain region to compute that average.

This type of analytical actions result in very demanding data access patterns, that if running on top of a transactional database can lock large amounts of data and consume computational resources in a way that could compromise the transactional system's availability. Ultimately, this could make it incapable of supporting the business transactions.

To relieve resource consumption, reduce operational risk in the transactional applications that support business and provide an optimized data structure for analytical cross-enterprise decision support purposes, Data Warehouses (DWs) are used. DWs clearly separate the analytical business processes from the transactional business processes. According to [13], we can assume the following distinct characteristics between transactional and analytical systems:

- From a perspective attending to its purpose, a DW is mainly a database system specifically designed for providing decision support information and business knowledge, while an operational system is specifically designed to support individual business transactions. Given that the business often requires the operational system to be online in order to accomplish a transaction, operational system requirements focus on enabling high availability to avoid risk in the accomplishment of the transactions themselves. On the other hand, since most decision support queries often require processing a large amount of data, DWs focus on fast query performance with high data throughput [13].
- From a perspective attending to the size and shape of its contents, a DW is composed of consolidated historical business data, mostly conformed within data schemas that optimize the execution of analytical queries. Generally, storing the business history implies taking up a very large amount of storage space, which often ranges from gigabytes to terabytes. In contrast, operational systems aim to keep their data sources "light", *i.e.*, small in size to minimize processing efforts and consequently keep their availability as high as possible. Transactional systems therefore keep only the exact amount of data which is required to support current and near-future business transactions.
- In what concerns their data schemas, transactional databases mostly have highly normalized schemas with a large number of tables and relationships amongst them, mainly to avoid data redundancy and keep each table small-sized, while DWs have denormalized schemas. Most DW database schemas are based on star schemas, where business facts are stored in a central table called fact table (*e.g.* sales table) and tables containing the business descriptors are called dimension tables (*e.g.* customer and product tables) [13]. Dimension tables link to the fact table by their primary keys (*e.g.* CustomerID, ProductID), are usually small in size (typically less than 10% of total storage space) and have a small amount of rows (up to tens of thousands), when compared with fact tables, which are typically very large in size and a huge amount of rows (millions or billions). Business facts are mainly stored in numerical-typed attributes within fact tables; since fact tables typically take up at least 90% of the DW total storage size [13].
- Considering his/her responsibility in the business, the DW user is typically a business manager or someone having a role of responsibility in the enterprise, while the typical user of operational systems are mainly transactional operators with low responsibility and few or none decision making privileges. Since they mainly consist of business managers and decision makers, the number of DW users it typically low (a few tens), while in many transactional systems the number of users is relatively high (tens to thousands).
- While end users of operational systems typically execute intensive read and write instructions, DW end users only execute read-only instructions, while DBAs and ETL (Extract-Transform-Load) users may insert or modify data. More than 90% of actions in DWs are typically analytical queries (*i.e.* SELECT statements), mainly executed against fact tables [13]. Reporting (*i.e.* periodically running reports for answering predefined decision support queries) is typical in DWs. Besides reporting, in many cases a very significant amount of decision support queries are *ad hoc*, which makes them mostly unpredictable in their syntax and frequency. In operational systems the queries are mainly simple, predefined and repetitive.

- Although analytical queries may typically access huge amounts of data, their response usually results in small datasets with a few hundred bytes and a relatively low number of columns (no more than a few tens). Most queries in DWs are CPU intensive and can take up to hours, while operational system queries are intended to be computationally fast and deliver very small response time.

Table 2 summarizes the main differences between operational systems and DWs, based on [13]. We shall now discuss how each type of ID technique is able to handle the characteristics inherent to each distinct type of database system.

4.2 Applying Intrusion Detection to Database Systems

As shown in Table 1, most DIDS focus on analyzing user command syntax (*i.e.*, parsing the SQL-expression syntax of queries to construct user profiles). As pointed out in [19], the most common problems with this type of approach are:

- Regular user queries may differ widely in syntax yet produce "normal" (*i.e.*, good non-intrusive) output, which generates false positives (*i.e.*, false alarms);
- Queries may be crafted by the attacker to differ slightly in syntax from the "normal" user behavior profiles yet produce "abnormal" (*i.e.*, malicious and intrusive) output, which generates false negatives (*i.e.*, attacks that pass undetected).

Given the expressiveness of the SQL language and the need to determine query equivalence or similarity, syntax analysis is complex and very difficult to perform correctly. In fact, query containment and equivalence is NP-complete for conjunctive queries and uncertain for queries involving negation [19].

In databases where typical user workloads have a well-defined number of distinct commands that are issued repetitively, relying on command syntax analysis may be feasible to achieve high ID efficiency. This is typically what occurs in transactional systems. However, in analytical systems such as DW's many actions are *ad hoc* and have variable execution times with variable data access patterns and dimension-size frequencies and thus, are mostly unpredictable and broad-scoped. This makes distinguishing between normal and abnormal commands in DWs an extremely difficult task. In such analytical databases, limiting ID to command syntax analysis by simply modeling SQL command templates or static frequent data access patterns (*e.g.* which tables or columns are accessed) is unreliable or, at least, minimalist.

Regarding the characteristics of DW user workloads, the ID solutions relying on temporal analysis such as presented in [18] are inadequate and mostly produce very poor ID results due to the unpredictable rate and execution time of those workloads. Due to the *ad hoc* nature of most of those workloads, ID solutions such as [2, 16] that are based on command template analysis lack the necessary dynamics to efficiently perform the ID processes and therefore also produce poor ID results. In transactional systems, temporal analysis is very efficient when the user actions occur within well-defined time periods and have predictable processing times. Otherwise, it suffers from the same issues with temporal analysis as those in DWs.

Table 2. Differences between Operational Systems and Data Warehouses

	Operational Systems	**Data Warehouses**
Workload nature/purpose	Transactional	Analytical
Temporal nature of the data	Current	Historical and current
Typical database storage size	As small as possible	Very large to huge
Typical number of tables	Medium to high	Small
Typical data schema type	Highly normalized	Denormalized
Typical number of users	Medium to large	Small
Typical user's business responsibility	Low	High
Typical type of command	Read/Write of small amounts of data	Read-only on large amounts of data
Typical command complexity	Simple	Medium to High
Typical operation dynamics	Static, predefined, predictable, repetitive	Reporting + Dynamic, *ad hoc*, iterative
Typical command response time	Small	Large
Typical command action	Read/write of a single row or few rows	Reporting and aggregation on many rows
Amount of data typically processed by each command	Small	Large or Very Large
Typical data update frequency	Often in a given period of time	Once periodically
Dataset size typically resulting from a command execution	Small	Variable (often Small)

Although the approach proposed in [19] adds a data-centric analysis of each user command execution's resulting dataset, the analysis is *a posteriori* to that execution. Given the time span between the start of the intrusion and its detection, together with resource consumption and sensitivity of the targeted data, many enterprises can suffer huge losses in case the intrusions compromise the system's availability or leak out business secrets, if their DIDS either takes too long to alert a malicious intrusion or is unable to prevent or stop its execution. In this sense, approaches *a posteriori* are not efficient solutions for ID in both transactional databases as well as DWs.

Conclusively, the unpredictable execution frequency and *ad hoc* nature of DW user workloads make time-based and SQL modeling ID approaches such as [2, 16, 18] mostly inadequate. Alternatively, DIDS performing ID at a coarse-grained basis such as database sessions or transaction command sets, instead of a fine-grained basis such as analyzing each SQL command, risk that a series of malicious commands may be executed before the intrusion can be dealt with. Therefore, data dependency and sequence alignment approaches such as [4, 33] that are able to inspect each user command *a priori* to its execution but only after a considerable amount of actions have been executed, should be carefully used according to each database context.

Data-centric techniques such as [19, 27] are capable of adding value to *a priori* ID techniques by executing an *a posteriori* analysis of the data affected by the user action. Combining these techniques with data access pattern analysis techniques such as [1, 11, 12], that deem the processed data, seem the most feasible and efficient DIDS for both types of database systems.

5. RESEARCH CHALLENGES

Considering the discussed issues, this section points out research challenges and guidelines for evaluating, developing and improving DIDS.

5.1 Intrusion Activity and Data Coverage

Command syntax-centric approaches focus on attack *syntax*, while data-centric approaches focus on its *semantics*. Distinguishing attack queries that have resulting datasets whose columns and resulting rows significantly differ from those of normal queries is covered by both syntax-centric and data-centric approaches, while data-centric approaches mostly capture attack queries that have similar columns but process or display different row contents from those of normal queries. Attack queries that are similar in both columns and resulting datasets are more easily discovered by syntax-centric approaches than by data-centric approaches.

To determine user intent, DIDS should not only focus on *how*, but also *what* data is *accessed* and *processed* (*i.e.*, which tables and columns, as well as which rows, are involved in the command's execution), and also *generated as a result* (*i.e.*, the resulting dataset's rows and columns). Besides the user command, the accessed and processed data and resulting datasets should be object of analysis. As far as we know, there is no DIDS approach mixing these items and covering this type of integrated broad-scope analysis.

5.2 Alert Management

Regardless of the ID technique, thresholds are typically used to define the probability of a certain action being an intrusion or compute if a particular alert should be considered significant or not. The main issue in these procedures is that high thresholds may allow many intrusions to pass undetected, while low thresholds may generate huge amounts of alerts, most of which probably refer to false alarms (alias false positives).

Given the sensitivity of data in many database systems, it is preferable to have low thresholds because the potential cost of non-detection is often too high or unacceptable. However, even slight changes to the parameters used in data mining and statistical DIDS may result in a huge, even exponential, increase of generated alerts. In this case, the number of false alarms is often so large that it leads to wasting immense time and resources, or the amount of alerts is so high that they are not possible to check in practice [25, 26, 28, 29]. This lowers the DIDS' efficiency and jeopardizes its feasibility, usefulness, and credibility.

Alert correlation techniques such as [6, 22, 25, 26, 31] have been proposed to deal with large amounts of generated alerts and decrease false positive rates, by grouping sets of alerts in order to apply some sort of classification that allows them to conclude which alerts are most probable of referring a true alert. Although they effectively reduce the number of alerts to check as well as the number of false alarms, we argue that they are not the best choice for alert management.

Since these techniques rely on filtering alerts, they may allow critical true intrusions – capable of producing a high amount of damage - to pass undetected, although they were initially flagged. We argue that no alert should be discarded and all alerts should be considered using alert ranking techniques instead, assessing the impact to the enterprise that might be caused by the intrusion to which the alert refers. Ranking the alerts improves damage or leakage containment by pointing out the intrusions that might cause more damage to the enterprise, so they can be rapidly dealt with.

5.3 Intrusion Impact Evaluation

To the best of our knowledge, there is no DIDS that evaluates the potential impact (*i.e.*, damage) that each potential intrusion is capable of doing to the database and/or enterprise. Given the business value of many database systems (*e.g.* data warehouses), this is a decisive issue to enable quickly dealing with threats representing high risk to the enterprise. Although approaches such as [28, 29] consider a measure of sensitivity for each attribute, no DIDS enables the assessment of the data itself that can be damaged or affected by the intrusion.

Besides detecting intrusions, DIDS should be able to measure or estimate a measure of the *impact* that could be produced by the intrusion. The main challenge is to determine *how sensitive* is the data targeted by the attack. Having the capability of measuring the impact of a given intrusion would allow classifying each intrusion action as tolerable or critical to the enterprise. As we previously mentioned, this could play a very important role in alert management in environments where large amounts of alerts are generated. Given that most alerts in these environments refer to false alarms, focusing on those that are in fact the most important ones would potentiate an efficient administration of intrusion response actions and resources.

5.4 Real-time Intrusion Detection, Response and Prevention Capabilities

Many DIDS execute the ID process *a posteriori*, *i.e.*, after the intrusion action has finished its execution, or are able to detect the intrusion while it occurs but are not able to stop it. Once again, given the value of data in many database systems and the potential costs of damage or information leakage, we consider the capability of a DIDS to detect and respond to intrusions in real-time as a critical requirement, *i.e.*, it must be capable of responding to an intrusion while it occurs and preferably before it produces any damage.

5.5 Evolution of ID Efficiency

DIDS should be able to automatically tune their ID algorithms in order to improve their efficiency by learning from their false positive and false negative results. They should enable calibrating their features, statistical functions and tests, classifiers and any other element belonging to the DIDS, and propose a method as how to achieve this. Such an approach for network ID has been proposed in [32], but as far as we know no similar solution has been proposed for automatically tuning DIDS. Machine learning techniques or other techniques that enable incrementally adjusting their ID parameters for improving their efficiency are advisable.

Another approach to improve DIDS is focusing on the typical characteristics of the database system in which they operate, *i.e.*, separating DIDS meant for operational systems from those meant for analytical systems. Given the distinct workloads between transactional and analytical databases, specifically tailored ID techniques for highly transactional environments as opposed to ID techniques for DWs should be able to achieve higher efficiency than those referred to as "all-in-one" general solutions.

5.6 Database Intrusion Detection Benchmarking

We acknowledge the fact that an experimental evaluation of a database intrusion workload setup using the ID techniques described in this paper would bring added value to this work, as well as support its discussion and conclusions. However, the datasets and attack loads used in database ID research are mostly synthetic and several came from proprietary real-world datasets, which makes them unusable for third parties. In fact, the only benchmark commonly used by several solutions was the KDD99 [5]; all the remaining used synthetically generated workloads and datasets or specific datasets from real-world scenarios.

This is the main reason why we do not discuss ID efficiency results from these publications. Furthermore, although the use of advanced techniques such as Support Vector Machines and artificial neural networks might suggest obtaining better results than simpler ones such as statistical measures, this is not clear or demonstrable at this point for the same reasons. Therefore, benchmarks are an essential instrument in the development and implementation of many systems. They are widely used for two main reasons:

- Benchmarks provide a mean to test those systems and supply solution providers and clients with measures that enables a meaningful comparison between different alternatives;
- They also provide relevant feedback to developers which enables them to improve those solutions.

Since the KDD99 benchmark focuses on intrusions at the network and operating system (OS) level, in what concerns databases a need arises for dealing with intruders that are able to bypass ID mechanisms working at the network and OS level. In spite of the criticality of protecting data against intrusions and the importance of having available benchmarks for testing and improving DIDS, there is no benchmark focusing on workloads at the database command level.

6. CONCLUSIONS

We have presented a survey on the available ID techniques and approaches used in DIDS and pointed out the issues that concern their usage. We argue that distinct database systems have unique user and data processing requirements that differ from each other and require distinctively tailored ID approaches. The difficulty in accurately profiling user behavior, the overstated number of alerts and false alarms generated by most ID techniques, the potentially low reliability on correlation techniques and the hypothesis that many intrusions may only be detected and dealt with *a posteriori* to the attacks and without any knowledge on the type of damage that the intrusions might produce, jeopardize the credibility and feasibility of DIDS in many specific high sensitive data contexts such as data warehousing environments [2, 25, 26].

Considering the typical very specific user workloads of distinct types of database systems and the importance of databases for enterprises, we conclude that specific DIDS should be developed, pursuing the following requirements and guidelines:

- Although role or session profiling can be used, a DIDS should be more accurate and efficient if it is able to manage individual user profiles;
- All user actions must be traceable, *i.e.*, the DIDS must be able to trace the user and IP address it comes from and the session to which it belongs;
- Perform real-time intrusion action analysis and have near real-time intrusion prevention and response mechanisms, as in [7, 12], preferably before each user command is executed;
- User action analysis for building profiles and executing ID must focus on fourfold items: user commands, accessed data, processed data, and execution results;
- Impact evaluation, *i.e.*, measuring the damage to the enterprise that might occur as a result of the intrusion action should be used for alert management for optimizing intrusion response;
- The ID techniques should be able to evolve through time, *i.e.*, they should be able to learn from each confirmed intrusion alert or false alarm and tune algorithms or models to increase detection rates and decrease false alarm rates;
- The availability and correct operation of the DIDS and the database must be mutually verifiable;
- When the DIDS is unable to prevent the execution of intrusive actions, it should consider the execution of a recovery-from-attack type solution, such as [3];
- Given the database performance issues in very large database systems such as DWs, the DIDS security aptitudes must seamlessly operate in settings with strict performance and scalability requirements.

To the best of our knowledge, no DIDS has been proposed that has been proved capable of efficiently complying with all the referred requirements and guidelines proposed in this paper.

Additionally, a DIDS benchmark for each type of database system should be developed and proposed by both the research community and industry. We acknowledge that defining benchmarks is not a trivial task and that there are always discussable issues concerning the objectivity and effectiveness of each proposal. A DIDS benchmark should provide a wide coverage of possible intrusion activity according to the several distinct user workloads, while simulating their execution in a realistic-like environment. Given the importance of ID in specific contexts such as DWs and the lack of standard benchmarks for testing DIDS at the SQL level, we believe that the issues presented in this paper are worthy of notice and hope that our work may motivate the discussion around the subject in both database and intrusion detection research communities, possibly driving the development of a standard benchmark for this purpose.

7. ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers of the ACM SIGMOD Record for their helpful comments. This paper was partially supported by project iCIS – Intelligent Computing in the Internet Services (CENTRO-07-ST24 – FEDER – 002003), Portugal.

8. REFERENCES

[1] Bertino, E., Kamra, A., Terzi, E. and A. Vakali. "Intrusion Detection in RBAC-Administered Databases", Annual Computer Security Applications Conference (ACSAC), 2005.

[2] Bockermann, C., Apel, M. and M. Meier, "Learning SQL for Database Intrusion Detection using Context-Sensitive Modeling", International Conference on Knowledge Discovery and Machine Learning (KDML), 2009.

[3] Chakraborty, A., Majumdar, A. K. and S. Sural, "A Column Dependency-Based Approach for Static and Dynamic Recovery of Databases from Malicious Transactions", International Journal of Information Security (9), 2010.

[4] Chung, C. Y., Gertz, M. and K. Levitt, "DEMIDS: A Misuse Detection System for Database Systems", IFIP TC11 WG11.5 Conf. on Integrity and Internal Control in Information Systems, Kluwer Academic Publishers, 1999.

[5] DARPA archive, Task Description of the KDD99 Benchmark, available at http://www.kdd.ics.uci.edu/databases/kddcup99/task.html.
[6] Debar, H., and A. Wespi, "Aggregation and Correlation of Intrusion-Detection Alerts", Recent Advances in Intrusion Detection (RAID), 2001.
[7] Dia, J., and H. Miao, "D_DIPS: An Intrusion Prevention System for Database Security", Int. Conference on Information and Communications Security (ICICS), 2005.
[8] Douligeris, C. and A. Mitrokotsa, "DDoS Attacks and Defense Mechanisms: Classification and State-of-the-Art", Int. Journal of Computer Networks (IJCN), Elsevier B. V., 44, 2004.
[9] Fonseca, J., Vieira, M. and H. Madeira, "Online Detection of Malicious Data Access using DBMS Auditing". ACM Int. Symposium on Applied Computing (SAC), 2008.
[10] Hu, Y. and B. Panda, "A Data Mining Approach for Database Intrusion Detection". ACM Intern. Symposium on Applied Computing (SAC), 2004.
[11] Kamra, A., Terzi, E. and E. Bertino, "Detecting Anomalous Access Patterns in Relational Databases". Springer VLDB Journal, 17, 2008.
[12] Kamra, A. and E. Bertino, "Design and Implementation of an Intrusion Response System for Relational Databases", IEEE Transactions on Knowledge and Data Engineering (TKDE), Vol. 23, No. 6, June 2011.
[13] Kimball, R. and M. Ross, The Data Warehouse Toolkit, 3rd Ed. Wiley & Sons, Inc., 2013.
[14] Kindy, D. A. and A. K. Pathan, "A Detailed Survey on Various Aspects of SQL Injection: Vulnerabilities, Innovative Attacks and Remedies", Int. Journal of Communication Networks and Information Security (IJCNIS), Vol. 5, No. 2, August 2013.
[15] Kundu, A., Sural, S. and A. K. Majumdar, "Database Intrusion Detection Using Sequence Alignment". International Journal of Information Security (9), 2010.
[16] Lee, S. Y., Low, W. L. and P. Y. Wong, "Learning Fingerprints for a Database Intrusion Detection System". Euro Symposium on Research in Computer Security (ESORICS), 2002.
[17] Lee, W. and D. Xiang, "Information-Theoretic Measures for Anomaly Detection", IEEE Symposium on Security and Privacy, 2001.
[18] Lee, V. C. S., Stankovic, J. A. and S. H. Son, "Intrusion Detection in Real-time Database Systems via Time Signatures". Real-time Technology and App. Symposium (RTAS), 2000.
[19] Mathew, S., Petropoulos, M., Ngo, H. Q. and S. Upadhyaya, "A Data-Centric Approach to Insider Attack Detection in Database Systems". International Conference on Recent Advances in Intrusion Detection (RAID), 2010.
[20] Motwani, R., Nabar, S. U. and D. Thomas, "Auditing SQL Queries", Int. Conf. on Data Engineering (ICDE), 2008.
[21] Newman, A. C., "Intrusion Detection and Security Auditing in Oracle". Application Security Inc. White Paper, 2011.
[22] Ning, P., Cui, Y. and D. S. Reeves, "Analyzing Intensive Intrusion Alerts via Correlation", Recent Advances in Int. Detection (RAID), 2002.
[23] Pei, J., Upadhyaya, S. J., Farooq, F. and V. Govindaraju, "Data Mining for Intrusion Detection: Techniques, Applications and Systems", Int. Conf. on Data Engineering (ICDE), 2004.
[24] Pham-Gia, T. and T. L. Hung, "The Mean and Median Absolute Deviations", International Journal on Mathematical and Computer Modelling", Vol. 34, Issues 7-8, October 2001.
[25] Pietraszek, T., "Using Adaptive Alert Classification to Reduce False Positives in Intrusion Detection". Int. Conf. on Recent Advances in Intrusion Detection (RAID), 2004.
[26] Pietraszek, T. and A. Tanner, "Data Mining and Machine Learning – Towards Reducing False Positives in Intrusion Detection". Inf. Security Technical Report, 10(3), 2005.
[27] Spalka, A. and J. Lehnhardt, "A Comprehensive Approach to Anomaly Detection in Relational Databases". IFIP Int. Conf. Data and Applications Security and Privacy (DBSec), 2005.
[28] Srivastava, A., Sural, S. and A. K. Majumdar, "Database Intrusion Detection using Weighted Sequence Mining". Journal of Computers, Vol. I, No. 4, 2006.
[29] Srivastava, A., Sural, S. and A. K. Majumdar, "Weighted Intra-Transactional Rule Mining for Database Intrusion Detection". Int. Pacific-Asia Conference on Knowledge Discovery in Databases (PAKDD), 2006.
[30] Treinen, J. and R. Thurimella, "A Framework for the Application of Association Rule Mining in Large Intrusion Detection Infrastructures", International Conference on Recent Advances in Intrusion Detection (RAID), 2006.
[31] Valdes, A. and K. Skinner, "Probabilistic Alert Correlation". International Conference on Recent Advances in Intrusion Detection (RAID), 2001.
[32] Yu, Z., Tsai, J. P. and T. Weigert, "An Automatically Tuning Intrusion Detection System". IEEE Transactions on Systems, Man, and Cybernetics, Vol. 37, No. 2, 2007.
[33] Zhong, Y. and X. Qin, "Database Intrusion Detection Based on User Query Frequent Itemsets Mining with Item Constraints", Information Security Conf. (InfoSecu), 2004.