Securing Data Warehouses from Web-Based Intrusions

Ricardo Jorge Santos[1], Jorge Bernardino[2], Marco Vieira[3] and Deolinda M. L. Rasteiro[4]

[1,3] CISUC – DEI – FCTUC – University of Coimbra – Coimbra, Portugal
[2] CISUC – DEIS – ISEC – Polytechnic Institute of Coimbra – Coimbra, Portugal
[4] DFM – ISEC – Polytechnic Institute of Coimbra – Coimbra, Portugal
[1] lionsoftware.ricardo@gmail.com, [2] jorge@isec.pt, [3] mveira@dei.uc.pt, [4] dml@isec.pt

Abstract. Decision support for 24/7 enterprises requires 24/7 available Data Warehouses (DWs). In this context, web-based connections to DWs are used by business management applications demanding continuous availability. Given that DWs store highly sensitive business data, a web-based connection provides a door for outside attackers and thus, creates a main security issue. Database Intrusion Detection Systems (DIDS) deal with intrusions in databases. However, given the distinct features of DW environments most DIDS either generate too many false alarms or too low intrusion detection rates. This paper proposes a real-time DIDS explicitly tailored for web-access DWs, functioning at the SQL command level as an extension of the DataBase Management System, using an SQL-like rule set and predefined checkups on well-defined DW features, which enable wide security coverage. We also propose a risk exposure method for ranking alerts which is much more effective than alert correlation techniques.

Keywords: Database security, Web security, Intrusion detection, Data warehouses.

1 Introduction

Many business models using web-based infrastructures require continuous access to decision support means such as Data Warehouses (DWs). To ensure this kind of access, DWs need to be available at any time from any location through communication infrastructures such as the Internet. This creates a main security issue, since it provides a mean for accessing their databases from outside the enterprise.

Intrusion is as a set of actions that attempt to violate the integrity, confidentiality or availability of a system [8]. Automatic detection of intrusion actions in databases is the main goal of Database Intrusion Detection Systems (DIDS). Since DWs are the core of enterprise sensitive data, quickly detecting and responding to intrusions is critical. However, most DIDS applied to DWs typically spawn too low true intrusion detection rates (*i.e.* false negatives) or too high false alarm rates (*i.e.* false positives) [7, 8, 9]. In the first case, many intrusions pass undetected; in the second case, the number of generated alerts is frequently so large that it leads to wasting vast amounts of time and limited resources, or they are simply just too much to be checked [7, 8]. This jeopardizes the credibility and feasibility of the DIDS [5, 7, 8]. Given the well-defined features intrinsic to DW environments, we argue they require specifically tailored DIDS. To the best of our knowledge, no such DIDS has been proposed.

Although alert correlation techniques [7, 10] have been proposed to decrease false positive rates, we also argue they are not the best choice for alert management in DW environments. These techniques filter alerts to determine those which present a higher probability of referring a true intrusion, given a predefined threshold. Using a threshold implies some alerts are discarded and thus, there is always the risk that a true intrusion may pass undetected. Given the value of DW data, this is not advisable. In our approach, we decide not to correlate/filter alerts, but measure their risk exposure (probability *vs* impact) to the enterprise. Instead of filtering alerts, our approach ranks all alerts by their potential cost to the enterprise, dealing with the most critical intrusions first instead of wasting time checking alerts with low impact for the enterprise.

The main achievements and contributions of our work are: Our DIDS is the first tailored for web-acessible DWs, analyzing each user command both *a priori* and *a posteriori* of its execution; It is also the first to use risk exposure to increase alert management efficiency; We use a very easy to understand and use declarative SQL-like form for defining rules at a fine-grain level for intrusion detection (ID) and response. Their flexibility covers a very large spectrum of possibilities that enables detecting and responding to a wide range of intrusions; The DIDS works as an extension of any DBMS, adding real-time ID and response management to the native database server; It can be easily implemented and used in any web-accessed DW, acting transparently at the application layer between DW user applications and the database.

The remainder of this paper is structured as follows. In section 3, we present our proposal, describing its architecture and each of its components and explaining how intrusion detection and response is managed. In section 4 we describe how each form of attack is dealt with by our solution. Section 5 presents related work on DIDS. Finally, in section 6 we present our conclusions and future work.

2 Data Warehouse Database Intrusion Detection System

Figure 1 shows the typical user action flow in a web-accessible DW, while Figure 2 shows the DIDS architecture, working as an extension of the DBMS.

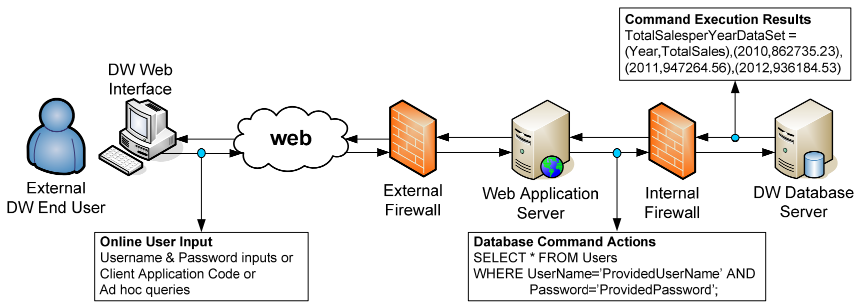
Command Execution Results
TotalSalesperYearDataSet = (Year,TotalSales),(2010,862735.23),(2011,947264.56),(2012,936184.53)
DW Web Interface
web
External DW End User
External Firewall
Web Application Server
Internal Firewall
DW Database Server
Online User Input
Username & Password inputs or Client Application Code or Ad hoc queries
Database Command Actions
SELECT * FROM Users WHERE UserName='ProvidedUserName' AND Password='ProvidedPassword';

Fig. 1. Typical user action flow in a web-accessible Data Warehouse

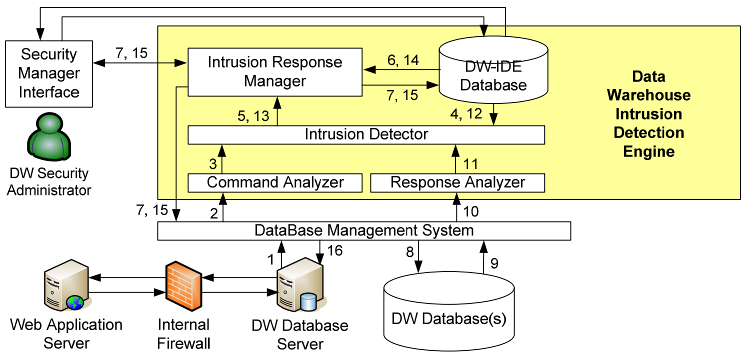

Fig. 2. The conceptual architecture of the DIDS for DWs

The sequence of intrusion detection steps is labeled in the figure and described as: A user requests an action through a *Web Application Server*, arriving at the DBMS for execution (step 1). Before executing it, the *Command Analyzer* retrieves the command text, date/time, and user/IP identification (step 2), parses the command, splits it into the ID features and passes all information to the *Intrusion Detector* (step 3). This component then gets the statistical model values for all features from the *DW-IDE Database* (step 4) and applies the ID algorithms (explained in subsection 2.2) to decide if the command is a potential intrusion. The detector then passes all information (features and respective intrusion detection results) to the *Intrusion Response Manager* (*IRM*) (step 5). Given each feature's result considering the user's action as an intrusion, the *IRM* retrieves the probability and impact rules, evaluate its risk exposure and generates the resulting alert (step 6), stores the data concerning the alert and the feature(s) that generated it in the *DW-IDE Database* for future reference, takes the appropriate actions to deal with the potential intrusion through the DBMS and notifies the *DW Security Administrator* (step 7). The *IRM* takes action by commiting the command's execution in the DBMS (in case it has been considered a non-intrusion) or by suspending or killing its execution, or killing the user session, either automatically or on request of the *DW Security Administrator* (step 7). If the user action is not considered an intrusion the *IRM* will simply update the feature's statistics in the *DW-IDE Database* (step 7) without notifying the *DW Security Administrator*.

If the *IRM* concludes the user's action is not an intrusion, it notifies the DBMS to normally execute it against the *DW Database(s)* (step 8). After the user command has been computed (step 9), its response is analyzed by the *Response Analyzer* before returning it to the interface which requested it (step 10), extracting the response features and passing them to the *Intrusion Detector* (step 11), which will repeat step 4 to detect possible intrusion action for each response feature. The *Intrusion Detector* will then pass the information to the *IRM*, which will repeat steps 5, 6, 7 as steps 13, 14 and 15. Finally, if the *IRM* concludes that the response is accepted or considered an intrusion, the computed results are respectively either sent back to the user interface which requested them or eliminated (step 16).

2.1 Risk Exposure Assessment

Given a user action, *risk exposure* is a function of both the *probability* it has of being an intrusion and the *impact* it may have, *i.e.*, the potential magnitude of the cost for the enterprise related to the damage or disclosure of the sensitive data which the action affects. Risk analysis consists on ranking the alerts given their computed risk exposure, according to a matrix similar to Table 1.

Table 1. The risk exposure matrix

		Probability			
		Very Low	**Low**	**High**	**Very High**
Impact	**Very High**	High	High	Very High	Critical
	High	Low	High	High	Very High
	Low	Very Low	Low	High	High
	Very Low	Very Low	Very Low	Low	High

To define which responses should be taken given the risk exposure matrix, the *DW Security Administrator* may define rules as the following:

```
GIVEN RISK EXPOSURE AS Low|Medium|High|Critical
ON FEATURE {FeatureName1, FeatureName2, ...}, AllFeatures
TAKE ACTION {DoNothing,Alert,PauseCommand,TerminateCommand,KillSession}
```

The definition of probability and impact rules that make up the assessment of risk exposure measures depend on the chosen intrusion detection features and sensitive data assessment by the DW Security Administrator, and will be explained in the next subsections. All risk exposure, probability and impact rules are stored in the *DW-IDE Database* and used by the *Intrusion Response Manager (IRM)*, as explained formerly.

2.2 Intrusion Detection and Response Management

Intrusion Detection. Given the distinctive assumptions for typical web-accessible DWs [4], in Table 2 we define the relevant ID features from a usability perspective. As shown, several features group values per user/IPAddress, other features are referred to values per command given each user/IPAddress, and further features refer those that are grouped by each session of each user/IPAddress. This allows testing features in different grouping levels (per user / per user session / per SQL command) and thus, widens the detection scope. Our approach adjusts a probabilistic distribution for each feature $\{F_1, \ldots, F_{29}\}$ for each user, from observations (feature values) during an initial training stage. To obtain those observations, we suppose the existence of an "intrusion-free" database command log. Executing that log's user commands we extract the values, *i.e.*, observations for building each feature's statistical distribution. Statistical adjustment tests are performed to obtain each population's distribution.

Table 2. Intrusion detection features

Features per User/IPAddress		
F#	**FeatureName**	**Description**
F_1	#ConsFailedLoginAttempts	The number of consecutive failed database login attempts by a UserID or from an IPAddress (accumulated or in a given timespan)
F_2	#SimultSQLSessions	The number of active simultaneous database connections
F_3	#UnauthorAccessAttempts	The number of consecutive user requests to execute an unauthorized actions (*e.g.* request to modify data when the database is read-only, or requesting to query data to which does not have access privileges)
	Features per User/IPAddress per Command	
F_4	CPUTime	CPU time spent by the DBMS to process the command
F_5	ResponseSize	Size (in bytes) of the result of the command's execution
F_6, F_7	#ResponseLines, #ResponseColumns	Nr. of lines and columns in the result of the command's execution
F_8, F_9	#ProcessedRows, #ProcessedColumns	Nr. of accessed rows and columns for processing the command
F_{10}	CommandLength	Number of characters
F_{11}	#GroupBy	Number of GROUP BY columns
F_{12}	#Union	Number of UNION clauses
$F_{13}…F_{17}$	#Sum, #Max, #Min, #Avg, #Count	Nr. of SUM, MAX, MIN, AVG and COUNT functions
F_{18}, F_{19}	#And, #Or	Nr. of AND and OR operators in the command's WHERE clause(s)
F_{20}	#LiteralValues	Nr. of literal values in the command's WHERE clause(s)
	Features per User/IPAddress per Session	
F_{21}	#GroupBy	Number of GROUPBY columns in all SELECT statements, p/ session
F_{22}	#Union	Number of UNION clauses in all SELECT statements, per session
$F_{23}…F_{27}$	#Sum, #Max, #Min, #Avg, #Count	Nr. of appearances of SUM, MAX, MIN, AVG and COUNT functions in all commands, per session
F_{28}	TimeBetwCommands	Time period (in seconds) between exec. of commands, per session
F_{29}	#SimultaneousCommands	Number of commands simultaneously executing, per session

For each active user session, we gather each new value generated for each feature and build sample sets. To detect an intrusion, statistical tests are performed: given each feature's original population, a new sample set is built joining that population with the user session sample set for that feature. New statistical tests are performed to adjust a new probability distribution to the former data collection. By testing if the new feature's distribution matches its original one (H_o), using Chi-square, Kolmogorov-Smirnov or Shapiro-Wilk tests, all performed at a level of 5% significance, for each test decision for a certain feature that results in rejecting the distribution's equality (H_o), we consider the user action as a probable intrusion.

Defining risk probability and impact. To determine each feature's individual importance in the overall intrusion detection process (which will be directly related to its risk probability), we attribute a weight to it. To compute its weight, we assume *a priori* each feature has the same relevance and will be incrementally self-calibrated using their respective True Positives *TP* (*i.e.* alerts generated by the feature that were confirmed as true intrusions) and False Positives *FP* (*i.e.* confirmed false alarms). For each feature F_i, its weight W_i is given by:

$$W_i = 0.5 + ((TP_i - FP_i) / (TP_i + FP_i)) / 2 \qquad (1)$$

where TP_i and FP_i are the total number of *TP* and *FP*, respectively, of all alerts generated by feature F_i. Every time an intrusion alert is generated by a given feature F_i, after it is checked the feature will have its *TP* or *FP* rate updated if it respectively refers to a true intrusion or a false alarm and, consequently, its weight W_i is also accordingly updated (increased or decreased). Thus, the self-calibrating formula works smoothly, giving a higher importance to the features that are more accurate.

To define the probability of each intrusion alert given the feature that generated it, our approach allows defining rules with the following syntax (list values with | are to be chosen from, while clauses in brackets are optional):

```
DEFINE PROBABILITY AS None|VeryLow|Low|High|VeryHigh
ON FEATURE {FeatureName1, FeatureName2, ...}, AllFeatures
[WHERE {List of filtering conditions}]
[WHEN {List of time-based conditions}]
```

Using this rule syntax, the intrusion probability of each feature F_i given its W_i as:

```
DEFINE PROBABILITY AS VeryLow ON FEATURE Fi WHERE Weight(Fi)<=0.25
DEFINE PROBABILITY AS Low
    ON FEATURE Fi WHERE Weight(Fi)>0.25 AND Weight(Fi)<=0.50
DEFINE PROBABILITY AS High
    ON FEATURE Fi WHERE Weight(Fi)>0.50 AND Weight(Fi)<=0.75
DEFINE PROBABILITY AS VeryHigh ON FEATURE Fi WHERE Weight(Fi)>0.75
```

The assessment of the impact caused by a user action is based on *which*, *how much*, and *when* sensitive data can be exposed or damaged by the user command, as well as *who* is the user. It is managed by using the following rules:

```
DEFINE IMPACT AS VeryLow|Low|High|VeryHigh
ON FEATURE {FeatureName1, FeatureName2, ...}, AllFeatures,
[WITH COLUMNS {Column1,Column2,...},AllColumns]
[WHERE {List of filtering conditions}]
[WHEN {List of time-based conditions}]
[JOINED WITH {Column1,Column2,...},AllColumns
```

The clauses are used in a similar manner to those in the probability rules, plus the clause distinguishing which is the user command (ON COMMAND) and the clause defining the impact of two or more columns being processed or shown together (WITH COLUMNS). The WHERE clauses in the DIDS rules (as in standard SQL WHERE clauses) allow a wide range of definitions and due to lack of space are not included. We just wish to make clear that the *IRM* algorithms can be easily adapted to cope with a wide range of rule possibilities, providing a very wide ID scope.

3 Experimental Evaluation

We used the TPC-H benchmark [18] to build a 1GB DW using Oracle 11g DBMS on a Pentium 2.8GHz machine with 2GB SDRAM (with 512MB dedicated to the database server), in a scenario with ten open web connections to the DW in which there are 2 "intruders" and 8 "true" DW users (non-intruders). For each "true" DW user's workload, a set of randomly chosen TPC-H benchmark queries were selected, *i.e.*, each user has different queries to execute, as well as a distinct number of queries. In each workload's queries, several were randomly picked for randomly modifying their parameters, to obtain a larger scope of diverse user actions. Each workload also included a random number of *random* queries (randomly picking a set of tables, columns, functions to execute, grouping and sorting, and literal restrictions for columns included in the WHERE clauses). The TPC-H queries represent typical reporting behavior, while *ad hoc* queries were simulated by random queries, in smaller number.

To build the statistical models for each feature of each "true" user, we executed each user's workload 50 times. To build each "intruder" workload, we generated 200 random intrusion queries of several types: SQL injection tautologies; Login/password guessing; inserting, changing or deleting a random number of rows; Selecting a random amount of columns and a random amount of functions (MAX, SUM, etc.) from a random number of tables, with and without a random number of grouping columns, with and without range value restrictions; SQL union queries with a random amount of columns and a random amount of tables; Query flooding; Unauthorized actions (create, drop, etc). These intrusion queries represent a wide variety of attacks.

The TPC-H benchmark has approximately seven years of business data. We consider the data from the most recent year to have high impact due to intrusion actions, the data from the two previous years as high impact, the data from the two years before that as low impact and the remaining as having very low impact.

Table 3 shows the ID results. Figure 3 shows the TP rate is considerably high (89%) while the FP rate is relatively low (5%), with an absolute number of 48 false alarms for a total of 225 generated alerts. Observing Table 4, the absolute number of false negatives is relatively low (23 in a total of 995 non-intrusions). The approach's precision is considerable (79%) and its accuracy is high (94%). Observing Table 4, the most relevant alerts (very high and critical) represent approximately one third of all alerts; these should be the ones first deserving attention on behalf of the security staff, instead of wasting time checking the remaining alerts (two thirds of all alerts), given that they present smaller impact. Finally, we measured an average overhead of 20% on user workload response time due to running the DIDS detection algorithms.

Table 3. Experimental results for the generated alerts (absolute values)

# True user actions	# intruder actions	#TP	#FP	#TN	#FN
1020	200	177	48	972	23

Table 4. Number of generated alerts per risk exposure measure

Very Low	Low	High	Very High	Critical	Total Number of Alerts
36	50	59	49	31	225

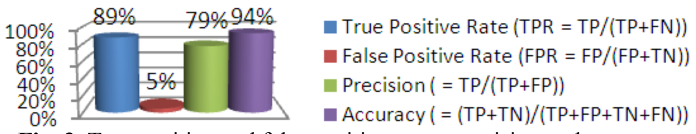

Fig. 3. True positive and false positive rates, precision and accuracy

4 Related Work

In [1], database transactions are defined by directed graphs describing the SQL command types used for malicious data access detection. This approach cannot handle *ad hoc* queries and works at the coarse-grained transaction level as opposed to a fine-grained query level. A Role Based Access Control mechanism for DIDS was proposed by [3]. Data mining techniques are used, namely classification and clustering, against SQL instructions stored in database audit files to deduce role profiles of normal user behavior. A limitation of this approach is that it cannot extract correlation among queries in transactions. Moreover, since this solution is role-based, it works at a higher coarse-grained level than the user-based profiling in our approach. Detecting attacks by comparison, summarizing SQL statements into compact access patterns named as fingerprints, is the focus of [5]. Profiling the data accessed by users to try to determine their intent is an approach used in [6], using statistical learning algorithms. They argue that analyzing what the user is looking for (*i.e.*, what data) instead of analyzing how s/he is looking for it (*i.e.*, which SQL expressions), is more efficient for anomaly detection. In our paper, we integrate both these views. Data correlation using data mining or machine learning techniques is used in [2, 7, 8, 10].

5 Conclusions and Future Work

We have pointed out issues involving ID in web-access DWs and proposed a specific DIDS for these environments. Our DIDS works transparently between user applications and the database server as an extension of the DBMS itself. The SQL-like rulebase allows extending DBMS data access policies and covers an extremely wide range of intrusion attacks. Risk exposure assessment is used for ranking and prioritizing the generated intrusion alerts, presenting clear advantages when compared with correlation techniques. Experimental results show our approach achieves high efficiency and accuracy for the tested setup. As future work, we intend to test our approach in real-world DWs, namely in cloud environments.

References

1. Fonseca, J., Vieira, M., and Madeira, H., "Online Detection of Malicious Data Access Using DBMS Auditing", ACM Symposium on Applied Computing (SAC), 2008.
2. Hu, Y., and Panda, B., "A Data Mining Approach for Database Intrusion Detection", ACM Symposium on Applied Computing (SAC), 2004.
3. Kamra, A., Terzi, E., and Bertino, E., "Detecting Anomalous Access Patterns in Relational Databases", Springer VLDB Journal, 17, 2008.
4. Kimball, R. and Ross, M. The Data Warehouse Toolkit. 2nd Ed, Wiley & Sons, Inc., 2002.
5. Lee, S. Y., Low, W. L., and Wong, P. Y., "Learning Fingerprints for a Database Intrusion Detection System", European Symp. on Research in Computer Security (ESORICS), 2002.
6. Mathew, S., Petropoulos, M., Ngo, H. Q., and Upadhyaya, S., "A Data-Centric Approach to Insider Attack Detection in Database Systems", Recent Advances in Intrusion Detection (RAID), 2010.
7. Pietraszek, T., "Using Adaptive Alert Classification to Reduce False Positives in Intrusion Detection", Recent Advances in Intrusion Detection (RAID), 2004.
8. Srivastava, A., Sural, S., and Majumdar, A. K., "Database Intrusion Detection using Weighted Sequence Mining", Journal of Computers, Vol. I, No. 4, 2006.
9. Treinen, J. J., and Thurimella, R., "A Framework for the Application of Association Rule Mining in Large Intrusion Detection Infrastructures", (RAID), 2006.
10. Valdes, A., and Skinner, K., "Probabilistic Alert Correlation", (RAID), 2001.
11. Transaction Processing Council, TPC Decision Support Benchmark H, www.tpc.org/tpch.