An Adaptive Intrusion Detection System using a Data Mining Approach

Sujaa Rani Mohan, E.K. Park, Yijie Han
University of Missouri, Kansas City
{srmhv7 | ekpark | hanyij }@umkc.edu

Abstract

Weak data dependencies in large databases coupled with poorly written web based applications are a major cause for malicious transactions. The problem of security becomes especially acute when access roles are changed among users. Also the poorly maintained data base caches are a cause for added security leaks. We propose an adaptive Intrusion detection system to keep track of the varying data dependencies as and when the definitions for various access roles are changed. We use an association rule based approach to track all relevant data dependency rule sets for different access roles using a hierarchical structure. We then identify malicious transactions from the transaction logs in the database using the data dependency rule sets. These rule sets are continuously updated and stored in a repository. Our approach is shown to reduce data access bottlenecks, and ensures minimal manual intervention for maintaining a secure database.

1. Introduction.

Securing important data from malicious users has been a long time concern for many both in the industry as well as in research. Nowadays with web applications used to access large databases over a network the need for Intrusion Detection has become a dire necessity. When a Database is first designed, it is designed and architected based on initial requirements obtained from the users of the proposed database. There are few security leaks and the web application is well written for the predicted database transactions. Usually a Database system so designed will not be expected to be very susceptible to intrusion. It is a well known fact that no software can be made completely bug free. Loop holes are usually over looked due to poor testing or oversight as part of the database designer. Also the database may require some re-definitions of the database access roles based on the changes in the user's tasks. New tables and views may have to be added or old ones removed which causes changes in the data dependencies among tables. Such changes are generally invoked to make the database more feasible and this sometimes drastically affects the security level. A once secure database now becomes a perfect haven for malicious attacks. This is the core problem that we are trying to solve in our paper.

A database which is a part of a network or a host is usually monitored by the database administrator. He defines the various access roles for the users. These users hence have restricted access. With a number of users accessing the database with usually common queries there is a high bottleneck that arises. To prevent slow access speeds, the result sets of some queries are cached in a database cache. This reduces the access time but also opens the door for malicious unauthorized accesses. Usually large enterprises have a lot of sensitive data and hence in most cases such caches are poorly used. Malicious activity also arises when access roles are changed or the permissions for a user are changed. Another way to intrude into the database is by performing an unauthorized sequence of transactions. For example, a delete operation on a data item cannot occur without reading the item first.

Intrusion Detection Systems(IDS) have been developed to identify any unauthorized attempts or successful attacks on any type of monitored data or resources available as part of a network or host system. Most IDSs detect such malicious activity either at the transaction level or at the operating system(OS) level [1]. It is also shown that transaction level attacks take care of most OS level attacks [2], [3]. But there are many attacks which occur internal to the network such as by a user with lesser privileges accessing data that requires more access rights. Such attacks can be identified by analyzing the transaction logs. A transaction log contains all the transactions made on a database. More on Transaction logs are explained in later sections. By analyzing these logs most malicious activity can be identified. In a typical database accessed over a network there may be as many as one million transactions a day and any kind of computational analysis will prove to be costly and tedious. There have been a number of approaches to reduce the time using different approaches, the most recent effective strategy being data mining [4].

Data mining is the analysis of data to establish relationships and identify hidden patterns of data which otherwise would go unnoticed. Even so existing approaches require analysis of millions of records. Our approach reduces the time to determine the sensitive data patterns from changing data access roles in the database, thereby identify any malicious activity and allow secure database caching at the network level.

Usually the database itself will not have security restrictions on individual data items. Access roles define the read/write/execute rights for each table in the database. Users generally query the database using transactions (or bulk transactions) through a web API. These transactions limit the number of valid queries that are allowed on a database. This is very common in web applications which use a large database over the internet. It is very easy to break into the database by writing malicious code to run illegal transactions and executing them on the database if the web application is not written carefully. In [5] the authors propose an effective means to locate and repair the damage on-the-fly for web based data intensive applications with reasonable (database) performance penalty.

In our approach we propose an adaptive IDS which defines a set of data dependency rule sets based on changing access roles which are maintained in a repository to identify such malicious transactions. The rest of the paper is organized as follows: Section 2 briefly discusses the current approaches and related work in this area of research. Section 3 outlines our concepts and assumptions used in our approach. We describe the various phases of our adaptive IDS in Section 4. We present an analysis of our approach to two well known database oriented web applications in Section 5. A brief conclusion and a discussion on our future work of applying an adaptive IDS to distributed databases using a multi agent framework is given in Section 6.

2. Current and related work.

Many researchers have dwelled into the field of database intrusion detection in databases using data mining. In [4], the author talks about a framework for continuously adapting the intrusion detection system for a computer environment as it is upgraded. The paper shows a number of data mining approaches to solve this problem and greatly discusses the results. Intrusion Detection has been approached using data mining by many researchers like [6]. In [5] , a multiphase damage confinement approach to ensure no damage is spread across the database after the detection is done. [7]'s paper uses a data dependency miner to identify correlations between data items and defines read and write sets for each data item. These rules are mined by scanning the database logs but it does not take into consideration the fact that the data dependency rules do not hold good for different access roles. We show in this paper that by applying the data dependency miner for transactions of each access role the, data dependencies will be more reliable. Also we show how the database transaction cache which caches frequently queried resultsets can be used more efficiently, by effectively preventing malicious accesses.

In [7], the authors argue that malicious writes are the major security threats and simple approach is given to identify malicious writes using data dependencies. The paper does not identify illegal reads. Our approach identifies all types of illegal accesses. Data mining is required to identify hidden data dependencies which might cause security threats and using triggers and stored procedures, we can only prevent expected loop holes. A possible solution is using dynamic stored procedures to maintain changing database data dependencies for intrusion detection but this an inflexible approach especially for a database which is accessed using a web based application.

Our approach deals with illegal transactions submitted to the DBMS via some mechanism using a user id with lower access rights using a web based application. Also it will prevent those intruders who bypassed the access control mechanism as the data dependency rule set will still track the valid sequence of reads and writes required for each transaction.

A change in the database models is inevitable and hence the security of the DB is at stake if the IDS model does not adapt itself to these changes. Also, adaptive database models have been shown to detect malicious transactions more effectively. In [4], the authors present a number of data mining approaches and their effectiveness for detecting malicious transactions.

3. Concepts and Assumptions

3.1. Database Schema, Access Roles and privileges.

A Database schema is a set of objects owned by a user. A user automatically has all object privileges for schema objects contained in his or her schema. A user can grant any object privilege on any schema object he or she owns to any other user or role. A privilege can be granted explicitly. For example, the privilege to insert records into table X can be explicitly granted to the user A. Alternatively a privilege can be granted to an access role which is a named group of privileges, and then the role can be granted to one or more users. For example, the privilege to insert records into a STUDENT table can be granted to the role named ADVISOR, which in turn can be granted to the users AdvisorB and ProfA.

In a relational Database such access roles are granted privileges in the form of a hierarchy with higher level access roles inheriting all privileges of lower level access roles. Our model works on this assumption. A sample hierarchical pattern is shown in Figure1

$A \leftarrow B \leftarrow C,D$

Figure 1

Each Access Role type is assigned a weight which represents its level in the hierarchy. For example from Fig.1, C and D will be at level 1, B at level 2 , and A at level 3. In our paper we define each type of access roles with a set of read/write permissions for each table or view.

3.2. Transaction.

A transaction consists of a sequence of reads and writes of different data items from different tables in the database. A transaction Tk can be denoted as < o1(d1), o2(d2), o3(d1) > where di where i = 1….n is the different data items in the database, oi is an operation of the data and belongs to the set of reads and writes <r,w>. The data sequence for a transaction can be shown as Dk = {d1,d2}. Such a transaction will exist for the read or write operation for every data item in the database. i.e to perform a read or write operation there maybe some other read or write operation required on other data items prior to this read/write. This is called data dependency between data items and such validations are usually not made in by the database. The database only checks for foreign key, and primary key dependencies. Our data mining analyzer analyses all transactions' read/write sequences and formulates data dependency rule sets that are valid for different access roles. From these rule sets we show in the following sections how malicious transactions can be identified.

3.3. Transaction logs.

Each transaction requested by a user is logged in the transaction log table. In our approach a log entry consists of the following fields:

i. Transaction ID
ii. Ischange (has the record changed as a result of the transaction)
iii. isDelete (has the record been deleted)
iv. isRestore (is this entry a restore point if the record is lost)
v. which data items have been changed
vi. isMalicious (is this transaction malicious)
vii. SecurityDegree (the minimum access role level required for the user to initiate such a transaction)
viii. UserId (who initiated this transaction)
ix. AccessRole (what access roles does this user have on the database)

In our approach, these logs show the malicious transactions and the degree of maliciousness.

4. Our Approach.

The Apriori algorithm has become a well known standard for identifying patterns using association rules [8]. Its main disadvantage is that, if a pattern of length n is needed, then n passes are needed through the items. This can become a large overhead for our current application. [9] describes an efficient approach to perform incremental Mining of Frequent Sequence Patterns in Web logs. The approach we have used in this paper is a variation of the Apriori algorithm [10] which identifies frequent rule sets using a pattern repository in linear time [11]. The main advantage of this approach is the ease of updating the rule set and scaling. New frequent rule sets added to the repository can be used immediately.

However, our problem requires rule sets which identify all relevant patterns and not only the frequent ones. The rule set should be complete and all dependencies that may cause a security threat need to be identified. Also our problem requires an algorithm that can quickly adjust the rule sets rather than completely redefining them depending on changing items and item sets. We use a pattern repository similar to [11] to keep track of valid data dependencies. Also we modify the Apriori algorithm to reflect all relevant data dependencies.

Our approach will help build frequent rule sets by analyzing all data dependencies for different user access role types to identify malicious activity from database transaction logs.

Our approach can be subdivided into the following phases.

4.1. Phase 1.

a) Initial Database Scan: This phase involves identifying the different tables, views, data items, primary key and foreign key constraints in the database.

b) Identifying Access Roles and their hierarchy: Based on the read/write/execute rights on different views and tables that each access role has the access roles are classified in a hierarchy with the access role having all rights (for example the database administrator) being at the top of the hierarchy. For example, Administrator $\leftarrow$ Professor $\leftarrow$ Student.

c) Let T be a universal Transaction set containing all read/write sequences for each transaction tx where x = 1….m , the total number of all transactions for the database based on all access role definitions. Let AR with the set of all access roles ranging from AR1 to ARn where ARn is the access role with all rights on the database.

d) Ordering Data items for rule set formulation: The ordering can be performed by performing a count on the number of times a data item occurs in all transactions in the universal transaction set T and then numbering the data items by the descending order of their counts. Also the primary key and foreign key constraints of the data items in a view or table should be taken into care while ordering as this ordering affects the order in which data items are selected while formulating the data dependency rule sets for the various access roles. This order will affect the formulation of rule sets involving data items that have cross dependencies between tables. This phase can only be partially automated and highly depends on the way the database is defined. For larger databases care should be taken not to assign cyclic dependencies between tables (for example, a write on Item x depends on a write in Item y and vice versa).

4.2. Phase 2.

a) We follow a reverse hierarchical order for rule set formulation. i.e. the rule sets for the access role with the least permission (lowest level in the hierarchy) are identified first followed by one of its siblings at the same level in the hierarchy as its access role definition will be different. Once all the rule sets for the lowest level in the hierarchy are identified, the rule sets for an access role in the next higher level are formulated and so on. All rule sets formulated for an access role are directly inherited by its parent and these rule sets are no longer reformulated.

b) For each access role chosen as per the reverse hierarchical order,

i. Let TAi{} be a subset of T containing the read/write operational sequences for all transactions for the access role ARi. Let each transaction be denoted as tz where z=1 to the number of transactions in TAi. Each transaction is a sequence of reads and writes, ordered from left to right in the order in which they need to occur. Let DAi be the set which maintains the counts for each unique aggregating rule set. The Access role ARi has operation-item set Ai{} which is initially empty.

ii. First pass: Determine all rule sets of length 1 (Length 1 means a single operation which may be a read or write on a data item). This is done by scanning the first operation in each transaction in TAi{} from left to right. Assign a count to the number of times each sequence of operation occurs in all TAi{} and store that count in DAi. These rule sets are now added to Ai{}.

Pass 2 to Pass MaxLengthTransaction of TAi{} or until bigger rule sets cannot be formed (Pass j): All Transactions which do not contain a rule set in Ai{} are removed from TAi{}. Determine all rule sets of length j. Repeat the same procedure as in Pass 1 and assign the count of the number of times the operation o(di) occurs with the existing rule set and store it in DAi.

By setting a support level we can remove infrequent rule sets from being formed at each pass by tracking the DAi counts for each aggregating rule set, but it must be noted that the purpose of this approach is to keep track of all relevant rule sets as opposed to frequent rule sets since the main aim is to identify all malicious transactions. A sample result set aggregation for a few passes for transactions starting with o(d1) is shown in Table1.

Table1. Sample data dependency rule set formulation.

Pass	Aggregating rule sets	DAi counts Cardinality of TAi = 10
1	o(d1)	8
2	o(d1) o(d2)	8
3	o(d1) o(d2) o(d3)	4
3	o(d1) o(d2) o(d4)	3
3	o(d1) o(d2) o(d5)	1
4	o(d1) o(d2) o(d3) o(d4)	4

Figure 2 shows the algorithm for the data dependency rule set formulation. Since the total number of iterations for both the FOR loops are less than the total number of transactions in the database, the algorithm runs with a predictable run time.

c) The rule sets that are in each access role's Ai{} are added to the rule set repository called the Data Dependency Repository (DDR). The DDR is kept up to date with all data dependency rule sets and heavily used to identify all malicious transactions and effectively secure the database cache as is explained in the following phases.

Figure 2 Algorithm for Rule Set Formulation

4.3. Phase 3

a) Identifying malicious Database transaction from Transaction logs: For every transaction a log entry is made into the Transaction logs. The IDS will identify the rule set for the transaction from the repository which will also give the minimum hierarchy level required to initiate this transaction. This hierarchy level shows the degree of maliciousness (whether a level 3 user is trying to access a database using level 5 access privileges).

b) Intrusion Trends: A performance check on the logs can identify trends in malicious transactions and by tracing the transactions marked malicious weak data dependencies which may cause these security leaks can be easily determined.

c) Securing database cache: Database caches which cache the resultsets from frequent queries to reduce bottle necks and database pool accesses, can now use the malicious degree assigned to the transaction to determine whether to cache the resultset or not. By not caching highly malicious resultset we can help prevent security leaks due to malicious accesses to the database cache.

```
1  i ← 0
2  For each ARi
3      Set Ai ← {}
4      Set TAi ←{ set of all transactions tz in T for ARi}
5          Set DAi ←{ set of all operations on all data items in TAi}
6      Set ruleSetLength ← 1
7          While(rule sets aggregate)
8          j ← 0
9          For each tz in TAi
10             add each unique sequence to Ai{}
11             DAi[j] ← number of times a unique operational (read/write) sequence of length ruleSetLength occurs in TAi's transactions. The transactions are scanned from left to right in that order only.
12             j ← j + 1
13         ruleSetLength ← ruleSetLength + 1
```

Figure 2. Algorithm for data dependency rule set formulation.

4.4. Phase 4.

Updating the DDR: In this phase all changes in the access role definitions are analyzed. (A mobile agent can be used to collect the changes from the database(s)). Not all rule sets need to be changed. Only that access role whose permission changed and its parents/grandparents need to have their rule sets updated as the other access roles will not be affected. It must also be noted that any change in a user's access role will automatically come into effect and will not affect the intrusion detection system as the rule sets will not change. Re-running the rule set formulation passes for every access role change is not necessary as generally all changes are made during database upgrades periodically by the database administrator during general maintenance. Hence re-running the rule set formulation will not decrease the performance of the database.

5. Analysis.

In this section we show how we have analyzed our approach by applying it to two well known scenarios thereby showing the effectiveness of the approach.
Scenario 1: Consider a typical Student-Course-Professor database. A sample definition for its access roles is shown in Table 2. The hierarchy can be represented as follows: Admin ← Professor ← Advisor, Student. This means that all rule sets that apply to the Advisor and/or Student roles will apply to the Professor role and all rights of the Professor, Advisor and Student are inherited by the Admin.

Table 2. Access role definition for scenario 1

	Course View	Student Info	Course Info	Professor Availability
Student	r	r	r	
Professor	r/w	r/w	r/w	r/w
Advisor	r	r/w	r	r/w
Admin	r/w	r/w	r/w	r/w

Scenario 2:
We simulated a typical Employee-Payroll Accountant-Employer database. A sample access role definition is shown in Table 3. The hierarchy can be shown as Admin ← Employer, Payroll Accountant, Employer.

Table 3. Access role definition for scenario 2

	Employee Personal View	Employees Payroll table	HR View
Payroll Accountant	r	r/w	r
Employer	r	r	r/w
Employee	r/w		r
Admin	r/w	r/w	r/w

Notice that in Scenario 2 all access role types except for the admin are at the same level. There are not many levels of hierarchy here and hence the algorithm took longer to generate all dependencies. In a typical system, the time taken to generate the initial rule sets will be the longest. Updating the rule sets in the repository will not affect the performance of the database by itself. It must be noted that the robustness of the result sets in identifying malicious transactions with changing access role definitions is the main factor determining the efficiency of this approach. For both scenarios: We used auto-generated log files of 10,000 random transactions from a list of valid and invalid transactions. Once Phase 2 is completed, a list of rule sets depicting data dependency patterns is stored in the repository and intrusion detection begins.

We then tested the rule sets with a list of 1000 random transactions consisting of both valid and invalid transactions. Every transaction is marked malicious by identifying the security degree for that transaction (The degree specifies the minimum user access role required to initiate that transaction) based on the access role the pattern adheres to. We changed the access role definition for Professors and the patterns were updated. The database slowed down significantly when the rule sets were reformulated every time an access role was redefined. We changed our setup so that all access role changes were collected over a period of time and the rule sets were updated only when a significant change in access roles has occurred (such that the support level for the rule sets is below the minimum or when its efficiency in identifying malicious transaction falls below the required level). In this case the reformulation can be planned and made to occur concurrently with database back ups. The transition down time for the databases IDS can thus be kept at a minimum when the rule set generator is rerun during off peak times or when the database is shut down for back up. On contrary all updates/deletes of users, user permissions did not affect the database performance and can be done on the fly as the rule sets are not affected by these changes.

The support level for including a rule set pattern was set to 25% which means the pattern must be seen in at least 25% of the total number of transactions considered. By varying the support level the security level changed accordingly. Care should be taken to set the support level as it directly affects the robustness of the rule sets in identifying malicious transactions.

The processing time for the rule sets is an initial cost. The updates do not interfere with normal database operations and the database is made more intrusion safe with negligible down time. Also the DDR helps in keeping the security leaks caused by poor database cache maintenance in check. Our data dependency rule set algorithm has been shown reduce computation time but since the analysis of the adaptive IDS was made on simulated values, our proposed approach has only been subjected to a preliminary testing. An actual deployment of the approach is currently being performed using intelligent agents and is discussed in the next section.

6. Conclusion and Future work

Our approach gives an efficient approach to deal with intrusion detection in large databases. Our adaptive IDS approach significantly reduces the computational time for identifying and maintaining valid rule sets using hierarchical access roles and pattern repositories. It significantly reduces the databases vulnerability to malicious transactions and weak data dependencies as a result of varying access role definitions. It provides an ability to detect malicious activity when it occurs within existing or past users of the database without slowing the database transactional activity.

Our approach can be deployed onto a real time database system in many ways. We are currently working on deploying our approach using light weight intelligent agents. Using light weight agents allows one to add more functionality in the future without affecting the performance of existing protocols.

Below are some of the agents that may be required to perform the various tasks in our approach:

a) An agent to scan and classify the database schema and access roles. It would take the role of a pre-processor during the initial set up and later to pre-process the transaction logs.

b) A mobile agent to identify changes in the access role definitions.

c) An agent to run the rule set formulation and update the Pattern repository. This agent is a static agent triggered by the mobile agent which identifies the access role definition itself.

d) A Database monitor agent is used to monitor the database caches.

e) All malicious transactions of all degrees can be monitored using a intrusion detection monitor agent.

These agents can be set to run every midnight (or when database activity is low) so that changes in the rule set will not decrease the IDS performance.

Our approach can also be scaled to distributed databases and mobile agents can be used to identify data dependency patterns across databases similarly. These agents are also light weight agents and we are currently working on the performance results for such an approach.

7. References.

[1] Peng Liu Jiwu Jing, Pramote Luenam, Ying Wang Lunquan Li, Supawadee Ingsriswang, “The Design and Implementation of a Self-Healing Database System”, School of Info Sciences and Technology Department of Information Systems, Pennsylvania State University UMBC, University Park, PA 16802 Baltimore, MD 21250.

[2 J. McDermott and D. Goldschlag, “Towards a model of storage jamming”, *Proceedings of the IEEE Computer Security Foundations Workshop*, Kenmare, Ireland, June 1996, pp. 176-185.

[3] Pramote Luenam, Peng Liu, “ODAM: An On-the-fly Damage Assessment and Repair System for Commercial Database Applications”, Dept. of Info. Systems, UMBC Baltimore, MD 21250.

[4] W. Lee, SJ Stolfo, KW Mok, “Data mining approaches for intrusion detection”, *Proceedings of the 7th USENIX Security Symposium*, 1998.

[5] P. Liu and S. Jajodia, “Multi-phase damage confinement in database systems for intrusion tolerance”, *Proceedings of the 14th IEEE Computer Security Foundations Workshop*, June 2001, pp. 191 – 205.

[6] Ashoka Savasere, Edward Omiecinski, Shamkant B. Navathe, “An Efficient Algorithm for Mining Association Rules in Large Databases”, *Proceedings of the 21st International Conference on Very Large Data Bases*, San Francisco, CA, USA, pp. 432 – 444, 1995.

[7] Yi Hu and Brajendra Prasad, “A Data Mining approach for Database Intrusion Detection”, *ACM Symposium on Applied Computing*, 2004, x(y): 711 – 716.

[8] Rakesh Agrawal, Andreas Arning, Toni Bollinger, Manish Mehta, John Shafer, Srikant Ramakrishnan, "The Quest Data Mining System", *Proc. of the 2nd Int'l ACM Conference on Knowledge Discovery in Databases and Data Mining*, Portland, Oregon, August 1996, pp. 244-249.

[9] Maged El-Sayed, Carolina Ruiz, and Elke A. Rundensteiner, "FS-Miner: Efficient and Incremental Mining of Frequent Sequence Patterns in Web logs", *Proc. of the ACM WIDM'04*, Washington, DC, November 2004, pp. 12-13.

[10] Rakesh Agrawal, Srikant Ramakrishnan, "Fast Algorithms for Mining Association Rules", *Proc. of the 20th Int'l ACM Conference on Very Large Databases*, Santiago, Chile, September 1994, pp. 487-499.

[11] Richard Relue and Xindong Wu, "Rule generation with the pattern repository", *Proc. of the IEEE International Conference on Artificial Intelligence Systems*, September 2002, pp. 186 – 191.