Detecting Intrusions in Databases

Asankhaya Sharma
Asankhaya@yahoo.com
NIT Warangal

Abstract

In this paper we propose a new architecture for database intrusion detection. Recently there has been considerable interest in the design of intrusion detection system for databases. Most of the current systems take a laid back approach and concentrate more on containment and recovery once the database has been infected by malicious transaction. We propose a more proactive solution (DIDAR); DIDAR aims to detect the intrusions as soon as possible with support for damage containment and auto recovery as well. DIDAR provides intrusion tolerance by working in two phases – learning and detection. During the learning phase based on the currently executing transactions we build a model of the legitimate queries for each user and later use that model to detect the malicious transactions. DIDAR guarantees quality of information assurance at 4 different levels for each user. We have positive results based on our prototype and preliminary testing on synthetic database. With almost no load to the database DIDAR achieves high detection rates, quick damage containment and full recovery.

Introduction

Intrusion detection is one of the prime areas of research currently in databases. With the advent of the internet and the World Wide Web, more and more work is now done online. All the necessary information from shopping orders to banking transactions is stored in databases. Protecting the information in such case is of utmost importance. Even with proper access control features there are several modes of attack possible like the SQL injection. Although proper coding practices can prevent most attacks there are still number of legacy programs that have to be protected. Moreover even in case of an attack the system should still be able to degrade gracefully. The most comprehensive system that addresses most of these problems is the Intrusion Tolerant Database (ITDB) [1]. The intrusion tolerant database system can operate through attacks in such a way that the system can continue delivering essential services in the face of attacks. With a focus on attacks by malicious transactions, it can detect intrusions, and locate and repair the damage caused by the intrusions [2].

There has been considerable amount of work done in detecting intrusions in databases. The use of data mining has been found useful in detection based on mining user query frequent item sets [3, 4]. Another approach is to fingerprint the transactions and then build a classifier system to differentiate between malicious and benign transactions [5]. The way the DIDAFIT [5] system fingerprints the queries is by building regular expression based models of the legitimate queries but we take an entirely different approach by using relations, attributes and conditionals of the query to construct a fingerprint. All these system involve a learning mechanism for model building in some form and hence have false positives. Our approach here tries to combine the benefits of

the data mining approach with the fingerprinting of transactions along with a feed back mechanism to give less false positives.

DIDAR Framework

The basic framework can be divided in two phases – learning and detection. During the learning phase we build a model of legitimate queries and use that model to identify malicious transactions in the detection phase. We give the detailed algorithms of each of the phase below.

1. **Learning Phase**

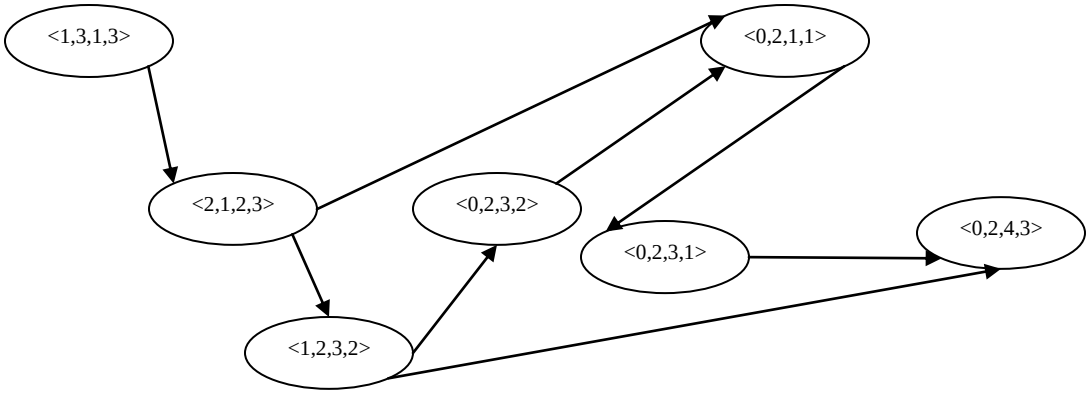
   During this phase the model of legitimate queries is built using supervised learning. We assume every transaction currently executing in the database to be benign. Any SQL query can be written as the following general form with three clauses.

   SELECT Attributes
   FROM Relations/Tables
   WHERE Conditions

   For every SQL query we associate a quadruple <t,R,A,C> which represents the fingerprint of the query [6].
   Where,

   't' stands for the type of query (SELECT, UPDATE or DELETE)
   'R' stands for the number of relations in the query
   'A' stands for the number of Attributes in the query
   'C' stands for the number of Conditions in the query

   Each such quadruple represents the whole query. Now for each user in the database we create a user access graph G (V, E) such that, V is the set of quadruples and E represent the access pattern of the queries in the database. While learning we read all the queries executing in the database, fingerprint them and convert them into a quadruple and add a node in the user access graph. Once the learning is finished the user access graph looks like something below.

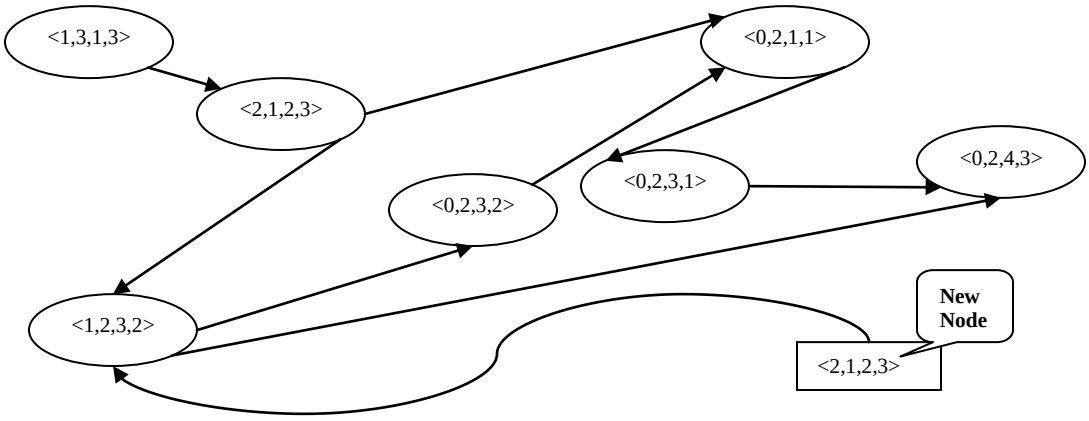


Once the learning is over each user has a user access graph where each node in the graph represents the fingerprint of the transaction. Based on this information we proceed to detection phase.

2. Detection Phase

Detection is fairly simple, each transaction that is currently executing is fingerprinted and converted into a quadruple. We traverse the user access graph and look for a matching node (say u) with same quadruple. If we cannot find such a node the transaction is labeled malicious or else we proceed again with the next transaction. Since we need to follow only the edges of the user access graph, for the next transaction we simply check all the nodes 'v' such that there is an edge between 'u' and 'v'. This way we can identify the malicious transactions.

Since it is not uncommon to have false positives we provide a feedback mechanism, if while in the detection phase some legitimate transaction is identified as malicious the user can give feedback and based on that we insert a new node in the user access graph with the quadruple representing the fingerprint of the current transaction. This will be clear from the following figure.



Once it is ensured the transaction is malicious we proceed according to the quality of information assurance (QoIA) [2] attached to the user.

Quality of Information Assurance

Different database users will have different needs and expect different levels of information assurance. So while protecting the database it doesn't make sense to provide one single level of security over the entire database. We propose four different levels of security which ensure quality of information assurance.

1. **Low**

   While the database is in a low level of security we only identify the intrusions with the feedback mechanism. There is no damage containment or recovery. This allows user to formulate a proper security perimeter with all possible transactions listed in the user access graph while also been aware of the security.

2. **Medium**

   In the medium level we provide the low level of security plus damage containment. After the detection we enter a damage containment phase.

   Damage Containment Phase

   During this phase we take a lock manually on all the tables accessed in the malicious transaction. By taking a lock we ensure that no other transaction can execute which can read data from the infected tables thus effectively containing the damage. As no new data can be infected this prevents the intrusion to cause damage spreading. The user can release the lock by rollback or commit the transaction after preparing for manual recovery.

3. **High**

   The key aspect of the high level of security is in addition to the medium level of security, even the recovery can be automated. Soon after the damage containment phase the recovery starts.

   Recovery Phase

   During automated recovery we rollback the database to the state just before the intrusion. Now we create a transaction dependency graph beginning from the malicious transaction. Using this graph we redo all the benign transactions. No malicious transactions are executed and hence the database heals itself to a correct and consistent state.

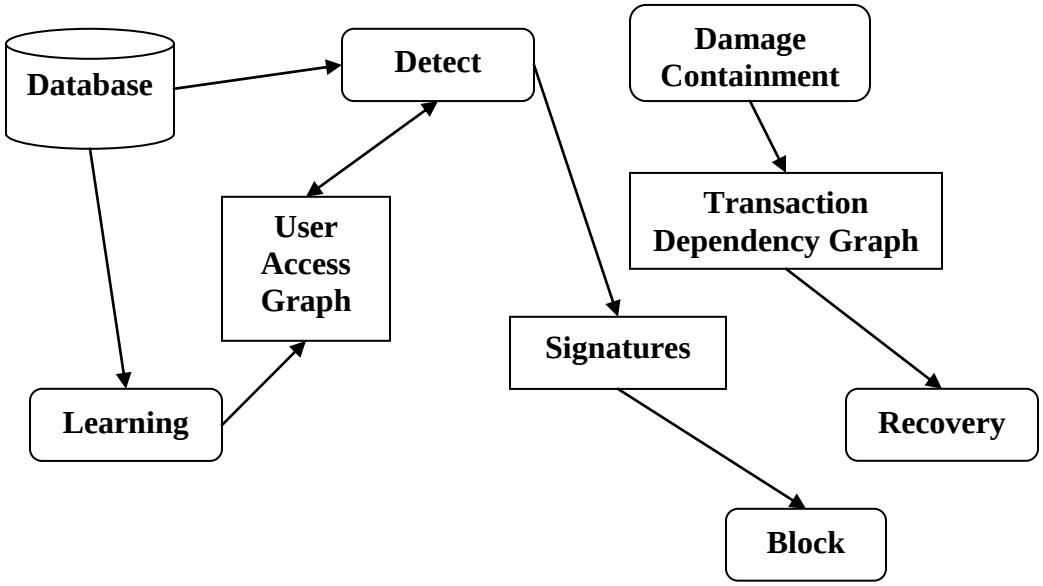
4. **Paranoid**

   This level provides the highest QoIA and uses most of the resources. We take the recovery one step ahead by introducing a blocking phase.

Blocking Phase

Once we have identified the transaction and done full recovery, in order to prevent such incident from happening in the future we calculate a signature of the transaction. This signature is used to identify the transaction even before we enter the detection phase. If the signature matches we directly lock all the tables in the query and effectively block the transaction. The detection and blocking phase go on together simultaneously; hence now with each user along with the user access graph, a list of signatures is also associated.

The following figure shows the most general representation of the DIDAR system.



Conclusions

We take a more proactive approach in detecting intrusions in a database. DIDAR has support for damage containment, auto recovery and signature based blocking of intrusions. The framework is comprehensive and provides intrusion tolerance while consuming minimum resources and low overhead to the database itself.

Future Work

The following are the directives for future work.

- Complete implementation of DIDAR based on a commercial database like oracle.
- More detailed fingerprinting for reducing false positives
- Building a transaction simulator to test the system under different conditions with synthetic and real data.

References

[1] Asankhaya Sharma, Govindarajan S, Srivatsan V, DIDAR - Database Intrusion Detection with Automated Recovery, B.Tech Thesis NITW 2007.

[2] Pramote Luenam, Peng Liu, The Design of an Adaptive Intrusion Tolerant Database System, Proceedings of the Foundations of Intrusion Tolerant Systems, 2003.

[3] Peng Liu, Architectures of Intrusion Tolerant Database Systems, Proceedings of 18th Annual Computer Security Applications Conference, 2002.

[4] Yi Hu, Brajendra Panda, A Data Mining Approach for Database Intrusion Detection, Proceedings of ACM Symposium on Applied Computing, 2004.

[5] Abhinav Srivastava, Shamik Sural, A.K. Majumdar, Database Intrusion Detection using Weighted Sequence Mining, Journal of Computers, vol. 1, no. 4, July, 2006.

[6] Wai Lup LOW, Joseph LEE, Peter TEOH, DIDAFIT detecting intrusions in databases through fingerprinting transactions, Proceedings of International Conference on Enterprise Information Systems, 2002.

[7] Bertino, E. Terzi, E. Kamra, A. Vakali, Intrusion Detection in RBAC-administered Databases, Proceedings of 21st Annual Computer Security Applications Conference, 2005.