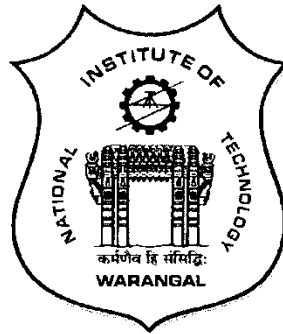


DIDAR – Database Intrusion Detection with Automated Recovery



*Submitted in partial fulfillment of the
requirements for the award of the degree of*

Bachelor of Technology (B.Tech)

By

Asankhaya Sharma (03712)

GovindaRajan.S (03723)

Srivatsan.V (03758)

Under the guidance of

Prof. D.V.L.N Somayajulu

**Department of Computer Science and Engineering
National Institute of Technology
(Deemed University)
Warangal (A.P.) – 506004**

April 2007

**Department of Computer Science and Engineering
National Institute of Technology
(Deemed University)
Warangal (A.P.) – 506004**



Certificate

This is to certify that, this is a bonafide record of the project work “**DIDAR – Database Intrusion Detection with Automated Recovery**” carried out by **Asankhaya Sharma (Roll no: 03712)**, **GovindaRajan.S (Roll no: 03723)** and **Srivatsan.V (Roll no: 03758)**, of Final Year B.Tech (Computer Science & Engineering), during the academic year *2006 – 07* in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology.

Prof. D.V.L.N Somayajulu
Department of CSE
National Institute of Technology
Warangal (A.P)

Dr. T. Ramesh
Head, Department of CSE
National Institute of Technology
Warangal (A.P.)

Acknowledgements

First, we would like to express our heart felt gratitude to our Project guide Prof. D.V.L.N Somayajulu (Faculty, Dept. of CSE), for his consistent guidance, encouragement and valuable suggestions through out the project span.

We would also like to thank Mr. K. Ramesh (Asst. Professor, Dept. of CSE), for his persistent suggestions. We would also like to owe the credit for this work to Mr. S. Ravichandra (Faculty In-charge, B.Tech Project Work, Dept of CSE) as well.

Finally, we would like to thank Prof. T. Ramesh, our Head of the department for providing us with this wonderful opportunity of doing this project work with complete support.

**Asankhaya Sharma
GovindaRajan.S
Srivatsan.V**

**(Final Yr B.Tech CSE)
NIT Warangal**

Abstract

In this project we present a new architecture for database intrusion detection. We implement this framework called DIDAR (Database Intrusion Detection with Automated Recovery) and discuss the performance issues. 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 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 we build a model of the legitimate queries for each user based on the currently executing transactions and later use that model to detect the malicious transactions. DIDAR guarantees quality of information assurance at four 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.

The implementation has been carried out in Visual C# 2005 and can run on any windows machine with .net framework 2.0. As for the database we choose oracle 10g express edition. The framework is general enough to be implemented for any commercial database system. To model a synthetic database we also built a transaction simulator which helps in testing and experimentation by injecting intrusions in the database.

Table of Contents

1. Introduction.....	5
1.1 Problem Specification.....	6
2. Related Work.....	7
3. Analysis.....	8
3.1 Learning Phase.....	9
3.2 Detection Phase.....	10
3.3 Isolation Phase.....	11
3.4 Recovery Phase.....	12
3.5 Blocking Phase.....	12
3.6 Data Warehousing and Data Mining Phases.....	12
3.7 Quality of Information Assurance (QoIA).....	13
Low.....	13
Medium.....	13
High.....	14
Paranoid.....	14
3.8 Granularity of Security Levels.....	14
Role Based Access Control.....	14
Context Based Access Control.....	15
3.8 Modeling.....	15
3.9 Specifications.....	16
Functionality.....	16
External Interfaces.....	16
Performance.....	16
Attributes.....	16
Design Constraints.....	17
4. Design Details.....	17
Architecture Diagrams.....	17
Learning Phase.....	17
Detection Phase.....	18
Data Warehousing Phase.....	19
Data Mining Phase.....	20
Class Diagrams.....	21
5. Implementation Details.....	28
6. Experimental Results.....	42
6.1 Test Cases.....	42
6.2 Test Results.....	43
7. Conclusions and Future Work.....	45
References.....	48
Appendix.....	50
List of Diagrams.....	50
Keywords.....	51

1. Introduction

The widespread use of the World Wide Web (WWW) has led to the constantly increasing use of web applications to store and share endless information. This information is generally stored in databases that are stable and generally robust. For example companies and organizations use Web applications to provide a broad range of services to users, such as on-line banking and shopping. Because the databases underlying Web applications often contain confidential information (e.g., customer and financial records), these applications are a frequent target for attacks. Thus there is a widespread need for an intrusion tolerant database to prevent such attacks against confidential data. 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. Our proposed intrusion detection and recovery system also ensures QoS (Quality of Service) which is offered to al the users who log in to the database as while protecting the database it doesn't make sense to provide one single level of security over the entire database

1.1 Problem Specification

To implement the proposed architecture of the DIDAR (Database Intrusion detection with automated recovery) system as an application with the following aims

1. To learn intrusions by creating fingerprints for all the executing transactions for each user and thereby creating user access graph for each user
2. To detect intrusions using the user access graph obtained from the learning phase and also using feedback mechanisms to assist detection.
3. To provide automated recovery and damage containment in case any malicious transaction (Intrusion) affects any part of the database.
4. To provide QoS (Quality of Service) for each user and to have different security levels for different users thereby adhering to the QoS provided.

2. Related Work

Intrusion detection started emerging as an interesting research topic since the beginning of the 1980s. In the 1990s, intrusion detection became an active area of research and even some commercial IDSs were built [7]. Over the last few years, several research works have been carried out that attempt to apply data mining for intrusion detection. Lee et al [8] suggest a method for network intrusion detection using data mining techniques. They consider classification, link analysis and sequential analysis as potential data mining algorithms along with their applicability in the field of intrusion detection. Barbara et al [9] have built a test bed for the detection of network intrusions using data mining. Though intrusion detection is comparatively a well researched field, only a few researches have considered the problem of database intrusion detection. Chung et al [10] use "working scope" to find frequent item sets, which are sets of features with certain values. They define a notion of distance measure that captures the closeness of a set of attributes with respect to the working scopes. Distance measures are used to guide the search for frequent item sets in the audit logs. Lee et al [11] have proposed real-time database intrusion detection using time signatures. It monitors the database behavior at the level of sensor transactions. Sensor transactions are usually small in size and have predefined semantics such as write only operations and well defined data access patterns. In real-time database systems, temporal data objects are used. This temporal data has to be updated periodically. If a transaction attempts to update a temporal data which has already been updated in that period, an alarm is raised.

Lee et al [10] suggest fingerprinting of the access patterns of genuine database transactions and using them to identify potential intrusions. They summarize SQL queries into compact and effective regular expression fingerprints. If a given query does not match any of the existing fingerprints, it is reported as malicious. Barbara et al [11] use hidden markov model (HMM) and time series to determine malicious data corruption. They build a database behavioral model using HMM that captures the changing behavior over time. Malicious patterns are the ones that are not recognized by the HMM with high probability. Zhong et al [12] use an algorithm that mines user profiles based on the

pattern of submitted queries. Hu et al [3] determine dependency among data items where data dependency refers to the access correlations among data items. These data dependencies are generated in the form of classification rules, i.e., before one data item is updated in the database, which other data items probably need to be read and after this data item is updated, which other data items are most likely to be updated by the same transactions. Transactions that do not follow any of the mined data dependency rules are marked as malicious transactions.

3. Analysis

The problem of intrusion detection was thoroughly analyzed. We consulted several papers and systems proposed by various investigators, most of which are available in the references. Based on the study carried out we propose a generic framework for database intrusion detection and recovery, we call it DIDAR which stands for database intrusion detection with automated recovery.

The basic DIDAR framework can be divided in several phases. These phases are listed below

- **Learning Phase**
- **Detection Phase**
- **Isolation Phase**
- **Recovery Phase**
- **Blocking Phase**
- **Data Warehousing Phase**
- **Data Mining Phase**

We give the detailed algorithms of each of the phase below

3.1 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 $\langle t, R, A, C \rangle$ which represents the fingerprint of the query [6].

Where,

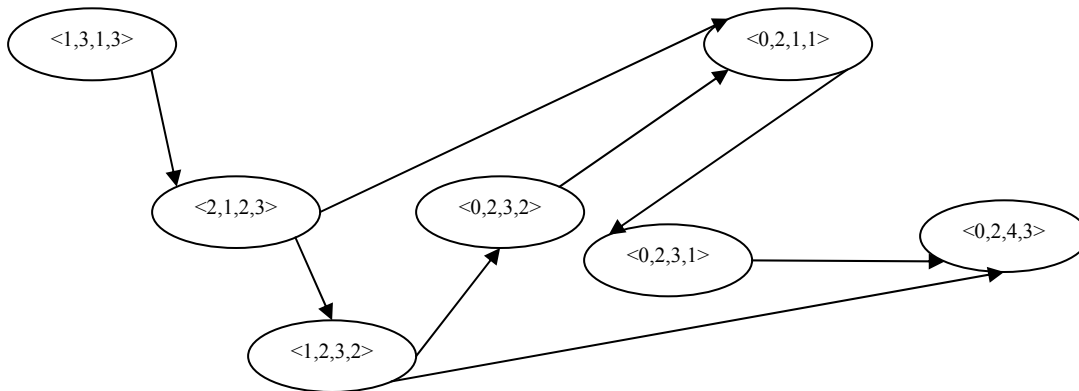
't' stands for the type of query (SELECT, UPDATE or DELETE)

'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.

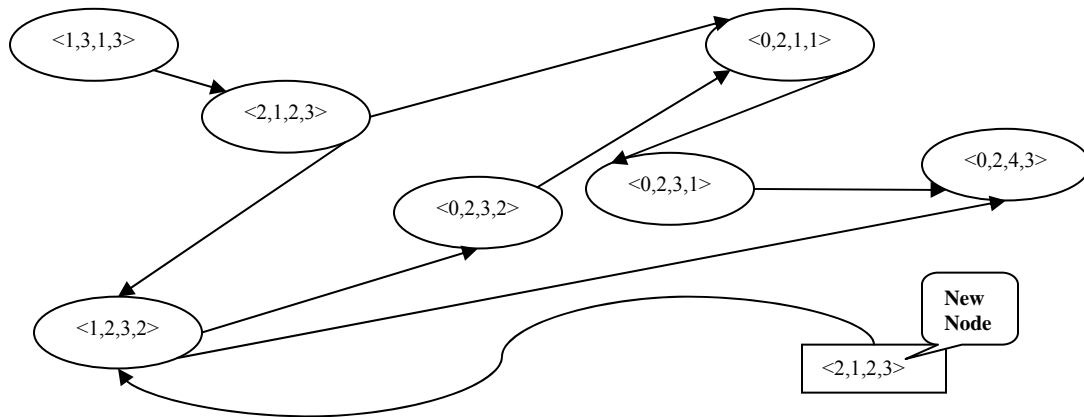


Once the learning is over each user has a user access graph where each node in the graph represents the fingerprint of the transaction. Based on this information we proceed to detection phase.

3.2 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. The other phases namely, the Isolation phase and the Recovery phase will be explained shortly to yield a better understanding of the entire architecture.

3.3 Isolation Phase

Once the malicious transactions have been identified in the detection phase then we need to do damage containment by isolating that particular transaction from others and prevent it from being executed. In the Isolation phase we make sure that such transactions do not execute or do not execute any transactions that depend on them. This is done by analyzing the transaction dependency graph for each such transaction and stopping any avalanche execution that is found from the graph. In other words we not only isolate the malicious transactions but also all transactions that depend on them, thereby preventing the intrusion from spreading across the database. The above diagram clearly illustrates that. From this phase the control then passes on to the recovery module (recovery phase) which will be addressed in the next section.

3.4 Recovery Phase

Once isolation of malicious transactions is done and their dependencies have been identified, the control then passes on to the recovery module which implements this phase. Here the recovery module's job is to undo any changes done by the malicious transactions to the database and restore the database back to the state where it was just before the execution of the malicious transaction. The recover model by itself may restore it back by either using some kind of log based recovery method or by using appropriate checkpoints that was created by the database. This is the last major phase of the implementation

3.5 Blocking Phase

This phase is used to block invalid/malign transactions from executing without actually having to go through the detection phase once again. The blocking phase works by initially building and associating a signature for each malicious transaction that is detected in the detection phase. Once that is done, then for each user in the database we will have a list of signatures also associated. Thus when a new transaction is executing for a particular user it is compared with the list of the existing signatures for that user and if there is a match the transaction is directly blocked without needing to go through the detection phase once again. Thus effectively the DIDAR system combines the advantages of the two traditional methods i.e. signature based approach and anomaly based approach. The following sections describe the two additional phases Data Warehousing Phase and Data Mining Phase.

3.6 Data Warehousing and Data Mining Phases

The most important thing during detection and recovery of intrusions is to decide upon the security levels for different users. Even though there are different ways of assigning security levels, the best would be from continuous monitoring of a user's accessing pattern etc. These phases accomplish exactly this. It's done by actually storing the user access patterns from the user access graph in a data warehouse at regular

intervals (say daily) (Data Warehousing Phase). Then from the data warehouse and the history of intrusions a classifier can be built for each user (Data Mining Phase). The security level can then be decided based on the classification and the attacks attempted on user data. Each of the intrusion is associated with a risk level which can be set by the user. This risk level forms the basis of classification of the security levels. We create a log of every successful intrusion detected, which has a risk level associated with it. This log can be lasted mined for information pertaining to the security levels at the discretion of the user.

3.7 Quality of Information Assurance (QoIA)

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.

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 being aware of the security issues related to the data being accessed and/or changed.

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.

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. During automated recovery we rollback the database to the state just before the intrusion actually took place. 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.

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 which has been discussed already in the previous section. The level is preferable only if the user executes transactions on critical or sensitive data whose integrity is of utmost importance.

3.8 Granularity of Security Levels

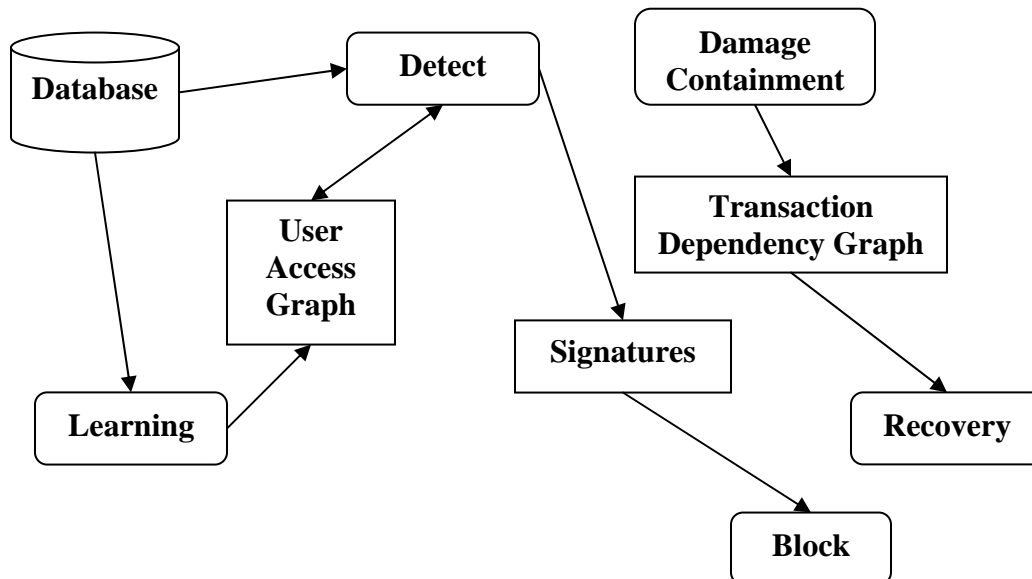
The problem of assigning different security levels to different users can be looked at a more detailed granular level by considering the following types of access controls.

Role Based Access Control - This actually is dictated by the user's access to the data and the different operations that he can do on it.

Context Based Access Control – The context of the user access i.e. the type of data being accessed, it's sensitivity etc. actually decide the level of security and access permissions for a particular user. These factors must also be taken into account before deciding on the security levels and access permissions for different users

3.8 Modeling

The following figure gives the information flow model of the DIDAR framework.



3.9 Specifications

Functionality

The software should provide the basic functionality of protecting a database in event of an intrusion. The database should remain live and online while the software works in the back ground. Software should not give too many alerts to the user and inform about only the relevant ones.

External Interfaces

The software interacts with the database as and when needed. This will ensure that the database is not unnecessarily overloaded with the queries originating from the software itself. The other external interfaces it can connect to is a data mining plug-in to decide on the security levels automatically.

Performance

The software should detect the intrusion in a reasonable time. The damage spread by the intrusion by the time it is successfully detected should be within the scope of repair by the recovery module. The software should not overload the database. Normal queries should execute in proper time even when the software is running. The recover time and detection time should be as small as possible.

Attributes

The software should be easily maintainable. We should be able to access the database thorough software from any terminal on the network and not only on the database server itself. The software should itself be secure from attacks that can compromise the database.

Design Constraints

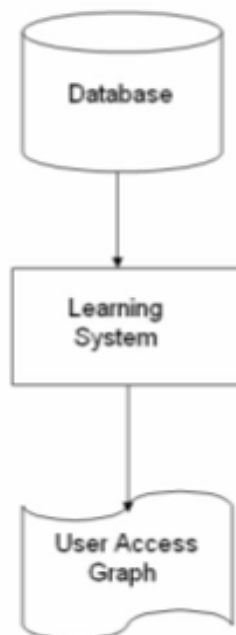
There are no major design constraints on the implementation. The software developed is in VC# 2005 and works for oracle 10g database. The IDE used for development is Microsoft Visual Studio 2005.

4.0 Design Details

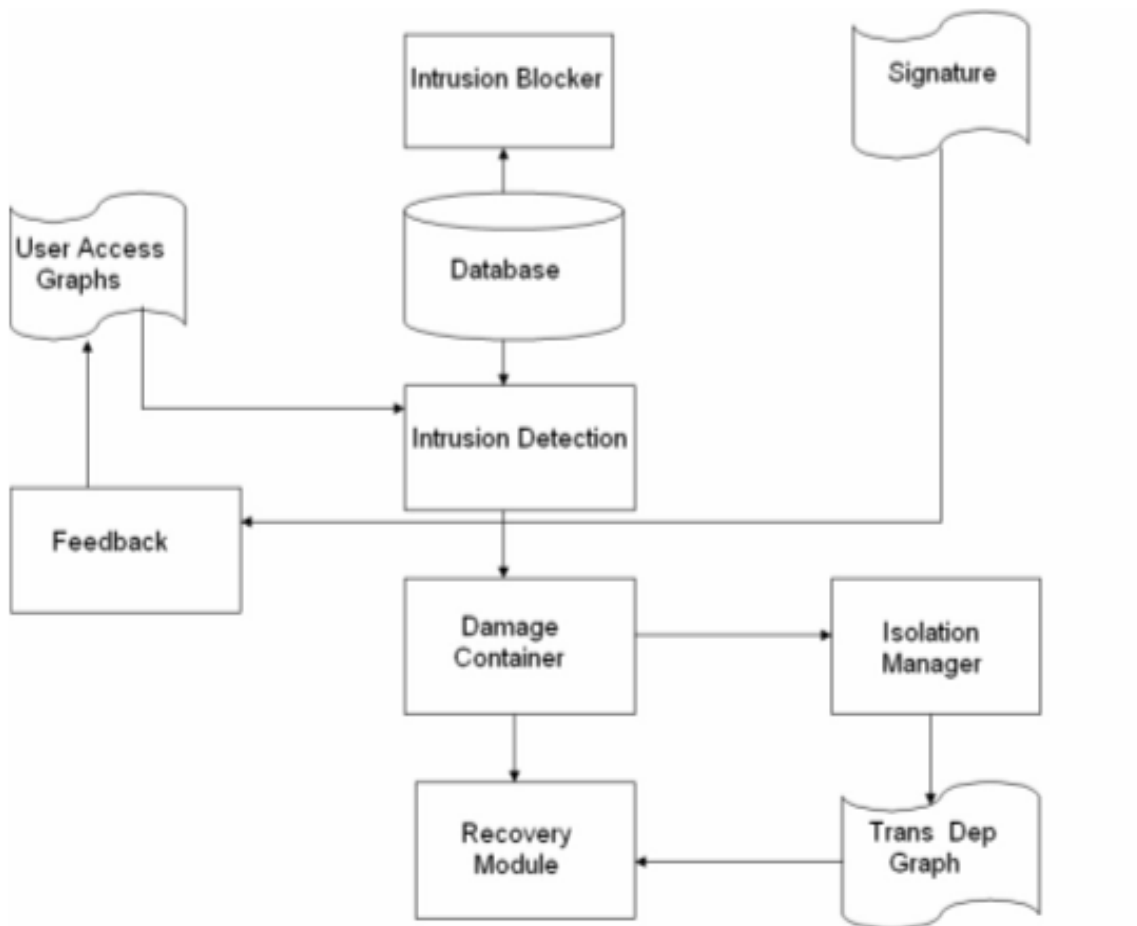
Architecture Diagrams

The following depicts the basic architecture of the various phases.

Learning Phase



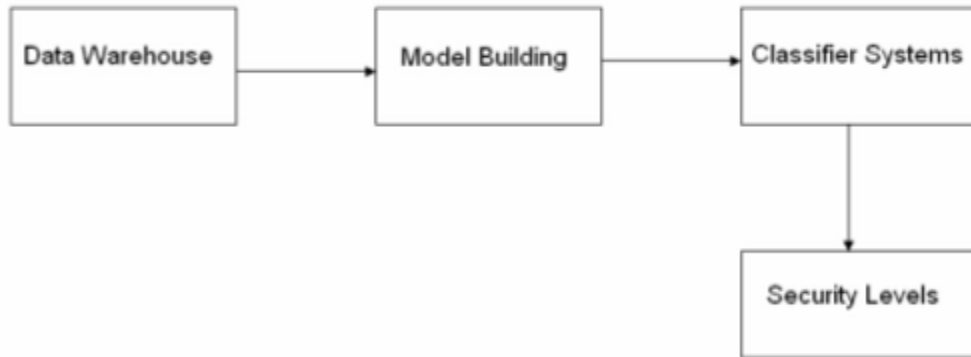
Detection Phase



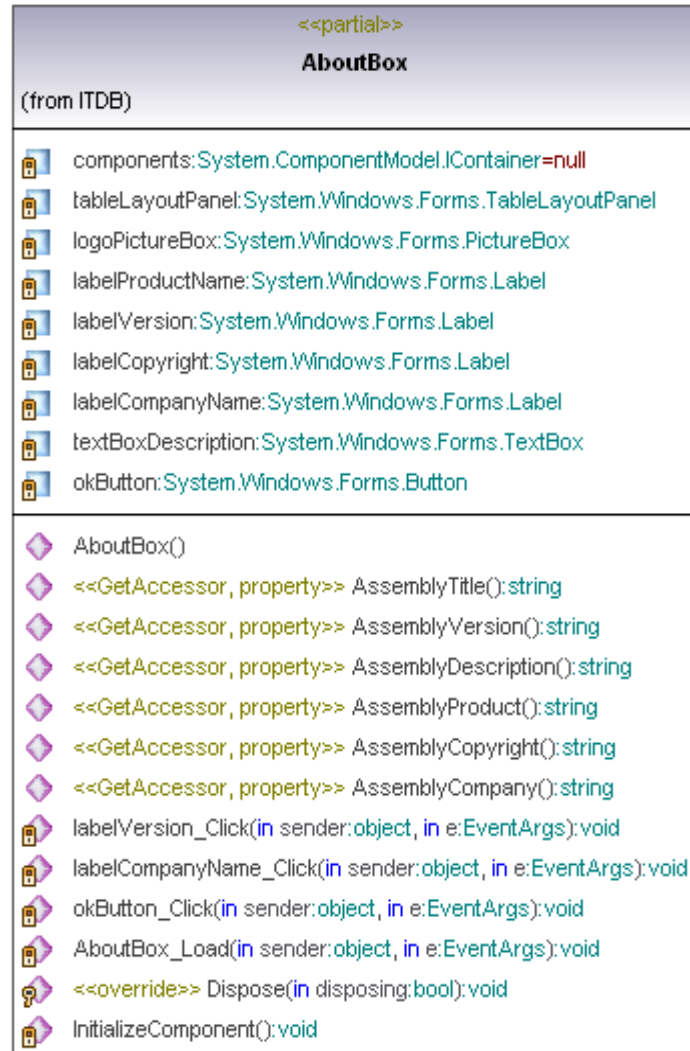
Data Warehousing Phase
























Data Mining Phase



















Class Diagrams







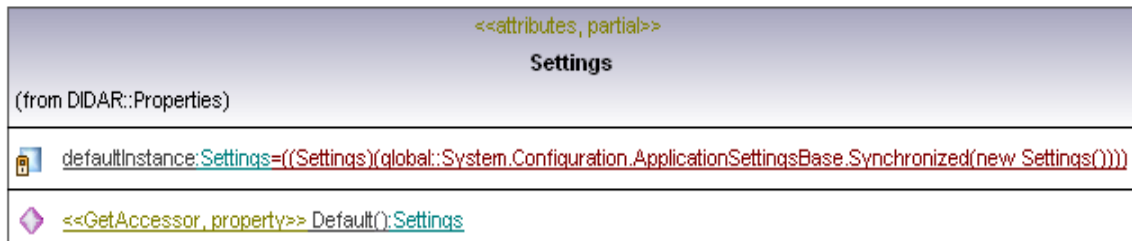
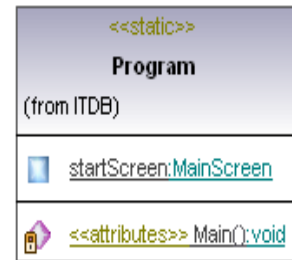
<<partial>>	
ConnectForm	
(from ITDB)	
	<code>user:String</code>
	<code>pass:String</code>
	<code>connstr:String</code>
	<code>user_name:String[*]</code>
	<code>sec_level:String[*]</code>
	<code>no_of_users:int=0</code>
	<code>components:System.ComponentModel.IContainer=null</code>
	<code>button1:System.Windows.Forms.Button</code>
	<code>button2:System.Windows.Forms.Button</code>
	<code>label1:System.Windows.Forms.Label</code>
	<code>label2:System.Windows.Forms.Label</code>
	<code>label3:System.Windows.Forms.Label</code>
	<code>textBoxd:System.Windows.Forms.TextBox</code>
	<code>textBoxu:System.Windows.Forms.TextBox</code>
	<code>textBoxp:System.Windows.Forms.TextBox</code>
	<code>ConnectForm()</code>
	<code>button2_Click(in sender:object, in e:EventArgs):void</code>
	<code>ConnectForm_Load(in sender:object, in e:EventArgs):void</code>
	<code>button1_Click(in sender:object, in e:EventArgs):void</code>
	<code><<override>> Dispose(in disposing:bool):void</code>
	<code>InitializeComponent():void</code>

<<partial>>
Form1

(from ITDB)

-  `usernames_query_exec:String[*]`
-  `noa:int=1`
-  `noc:int=0`
-  `nor:int=1`
-  `querytext:String=""`
-  `prev_query:String=""`
-  `cmdtype:String`
-  `qt:String[*]=new String[100]`
-  `index:int=0`
-  `<<const>> space:char=' '`
-  `table_names:String=""`
-  `table_count:int=0`
-  `components:System.ComponentModel.IContainer=null`
-  `query_exec:System.Windows.Forms.TextBox`
-  `label1:System.Windows.Forms.Label`
-  `timer1:System.Windows.Forms.Timer`

-  `Form1()`
-  `timer1_Tick(in sender:object, in e:EventArgs):void`
-  `<<override>> Dispose(in disposing:bool):void`
-  `InitializeComponent():void`



```
<<partial>>
Settings
(from ITDB)

interval:int=5000
risk_high:String="6"
risk_medium:String="4"
risk_low:String="2"
prev_high:String
prev_med:String
prev_low:String
components:System.ComponentModel.IContainer=null
label1:System.Windows.Forms.Label
textBox1:System.Windows.Forms.TextBox
groupBox1:System.Windows.Forms.GroupBox
label2:System.Windows.Forms.Label
groupBox2:System.Windows.Forms.GroupBox
label3:System.Windows.Forms.Label
button1:System.Windows.Forms.Button
button2:System.Windows.Forms.Button
textBox2:System.Windows.Forms.TextBox
label6:System.Windows.Forms.Label
label5:System.Windows.Forms.Label
label4:System.Windows.Forms.Label
button3:System.Windows.Forms.Button
textBox4:System.Windows.Forms.TextBox
textBox3:System.Windows.Forms.TextBox
label7:System.Windows.Forms.Label

Settings()
textBox1_TextChanged(in sender:object, in e:EventArgs):void
button1_Click(in sender:object, in e:EventArgs):void
button2_Click(in sender:object, in e:EventArgs):void
button3_Click(in sender:object, in e:EventArgs):void
<<override>> Dispose(in disposing:bool):void
```

<<partial>>

MainScreen

(from ITDB)

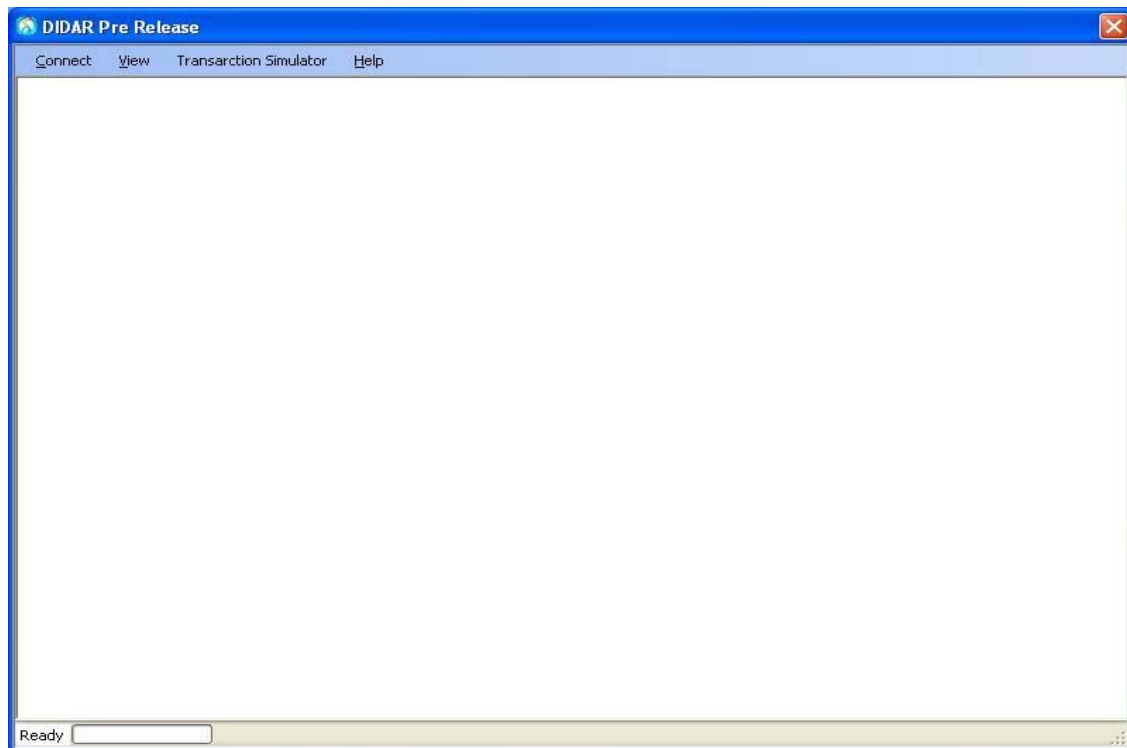
-  closeOnCtrlBox: Boolean=false
-  trans_simu: String="Turn Off"
-  ora: String
-  selected_mode: String
-  querytext: String=""
-  prev_query: String=""
-  cmd_type: int
-  no_of_rel: int
-  no_of_attr: int
-  no_of_cond: int
-  max: int=20
-  random: Random=new Random()
-  fingerprint_dts: String[*]
-  fingerprint_lts: String[*]
-  fingerprint_d: String[*]
-  fingerprint_l: String[*]
-  fingergraph: String[*][*]
-  user_current: String=""
-  sql: String
-  user_index: int=0
-  is_blocked: int[*]
-  table_names: String=""
-  table_count: int=0
-  noa: int=1
-  noc: int=0
-  nor: int=1
-  block_file: int=0
-  fwriter2: StreamWriter
-  timer_interval: int=0
-  components: System.ComponentModel.IContainer=null
-  topmenuStrip: System.Windows.Forms.MenuStrip
-  connectToolStripMenuItem: System.Windows.Forms.ToolStripItemMenuItem
-  databaseToolStripMenuItem: System.Windows.Forms.ToolStripItemMenuItem
-  viewToolStripMenuItem: System.Windows.Forms.ToolStripItemMenuItem

- viewToolStripMenuItem: System.Windows.Forms.ToolStripItem
- optionsToolStripMenuItem: System.Windows.Forms.ToolStripItem
- helpToolStripMenuItem: System.Windows.Forms.ToolStripItem
- aboutToolStripMenuItem: System.Windows.Forms.ToolStripItem
- exitToolStripMenuItem: System.Windows.Forms.ToolStripItem
- statusStrip: System.Windows.Forms.StatusStrip
- progressbar: System.Windows.Forms.ToolStripProgressBar
- Status: System.Windows.Forms.ToolStripStatusLabel
- notifyIcon: System.Windows.Forms.NotifyIcon
- notifyContextMenuStrip: System.Windows.Forms.ContextMenuStrip
- toolStripMenuItem1: System.Windows.Forms.ToolStripItem
- showToolStripMenuItem: System.Windows.Forms.ToolStripItem
- arrLabels: System.Windows.Forms.Label[*]
- arrLabels2: System.Windows.Forms.Label[*]
- arrButtons: System.Windows.Forms.Button[*]
- transactionSimulatorToolStripMenuItem: System.Windows.Forms.ToolStripItem
- label1: System.Windows.Forms.Label
- arrcomboBox: System.Windows.Forms.ComboBox[*]
- spyTheDatabaseToolStripMenuItem: System.Windows.Forms.ToolStripItem
- optionsToolStripMenuItem1: System.Windows.Forms.ToolStripItem
- turnOnToolStripMenuItem: System.Windows.Forms.ToolStripItem
- turnOffToolStripMenuItem: System.Windows.Forms.ToolStripItem
- timer2: System.Windows.Forms.Timer
- button1: System.Windows.Forms.Button
- settingsToolStripMenuItem: System.Windows.Forms.ToolStripItem

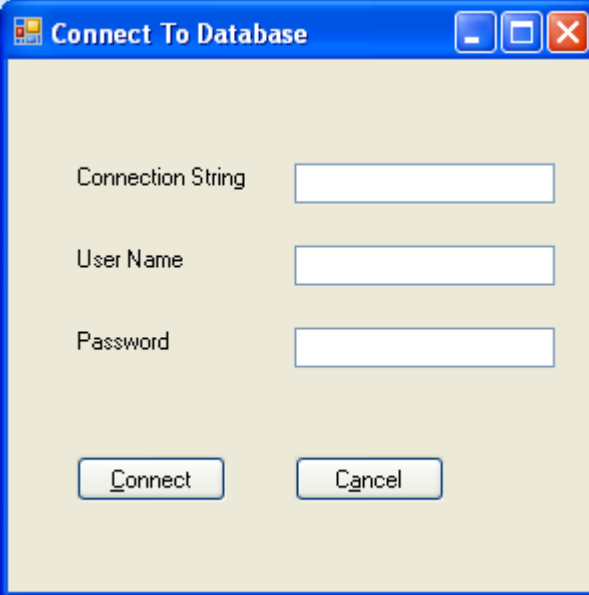
- ◆ MainScreen()
- MainScreen_Load(in sender:object, in e:EventArgs):void
- databaseToolStripMenuItem_Click(in sender:object, in e:EventArgs):void
- exitToolStripMenuItem_Click(in sender:object, in e:EventArgs):void
- toolStripMenuItem1_Click(in sender:object, in e:EventArgs):void
- aboutToolStripMenuItem_Click(in sender:object, in e:EventArgs):void
- optionsToolStripMenuItem_Click(in sender:object, in e:EventArgs):void
- DBUsers_SelectedIndexChanged(in sender:object, in e:EventArgs):void
- <<override>> OnClosing(in e:CancelEventArgs):void
- toolStripMenuItem1_Click_1(in sender:object, in e:EventArgs):void
- spyTheDatabaseToolStripMenuItem_Click(in sender:object, in e:EventArgs):void
- turnOnToolStripMenuItem_Click(in sender:object, in e:EventArgs):void
- ◆ detect_trans_simu():void
- ◆ learn_trans_simu():void
- turnOffToolStripMenuItem_Click(in sender:object, in e:EventArgs):void
- button1_Click(in sender:object, in e:EventArgs):void
- timer2_Tick(in sender:object, in e:EventArgs):void
- settingsToolStripMenuItem_Click(in sender:object, in e:EventArgs):void
- ◆ learn(in temp:String, in user:String, in sec:String, in ind:int):void
- ◆ detect(in temp:String, in user_current:String, in sec:String, in tablename:String, in user_index:int, in table_count:int):void
- ◆ lock_tables(in tbl_names:String):void
- <<override>> Dispose(in disposing:bool):void
- InitializeComponent():void

5. Implementation Details

As mentioned already the implementation of the DIDAR system is done and constrained well within the scope and restrictions available to us. This section illustrates the use and the choice of the various parameters involved in the implementation of DIDAR. The system is implemented in VC# 2005 edition and is available to work on Oracle 10g edition. The main idea of detection of malicious transactions is carried out in a single class which is the MainScreen. The following figure illustrates the user interface part of the MainScreen.cs program.



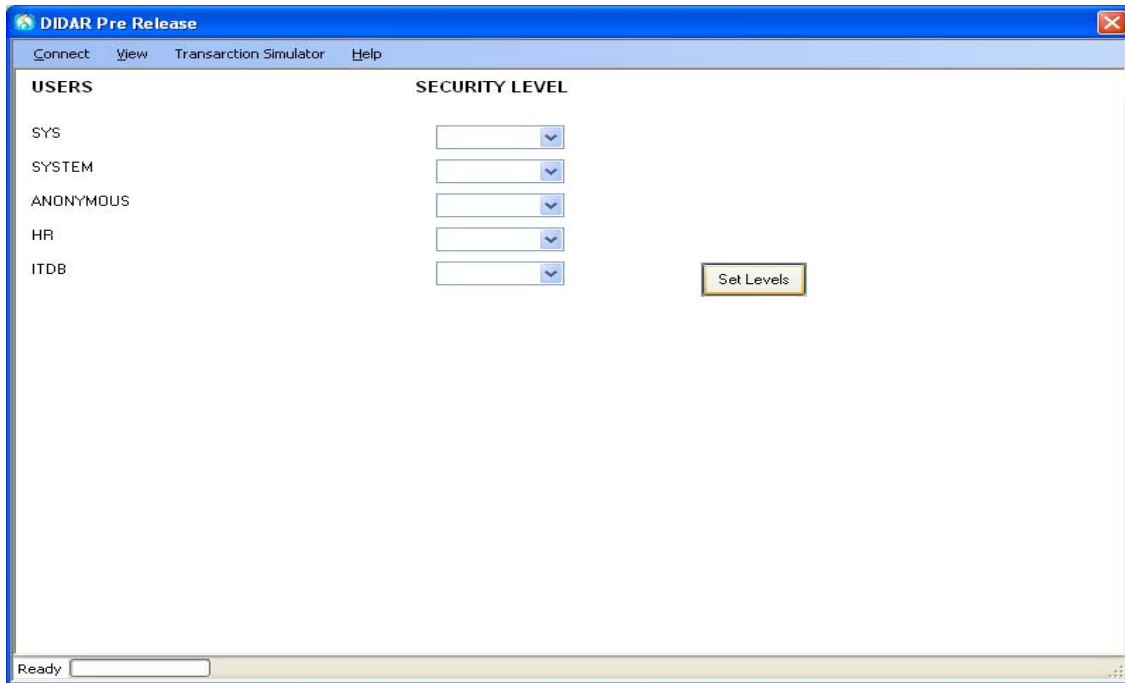
The Connect menu in this MainScreen offers the user to either connect to a database or to exit the application. On clicking connect the ConnectForm opens up asking the user to enter the username and the password for connecting to the database.



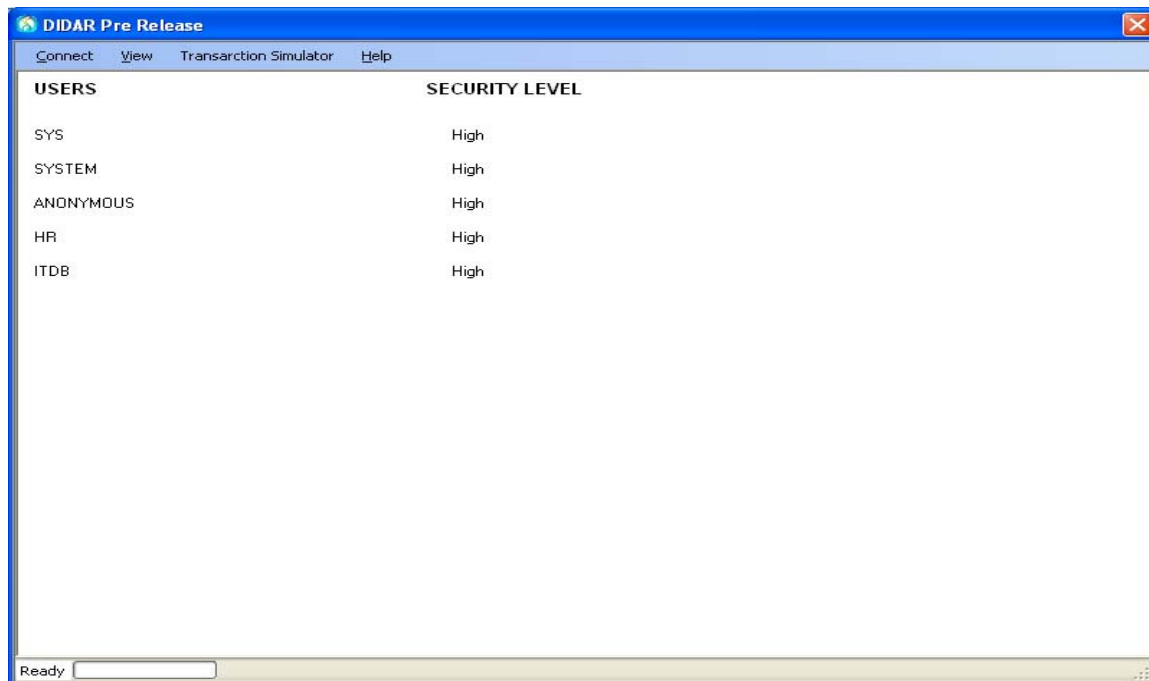
The image shows a standard Windows-style dialog box titled "Connect To Database". It features a blue title bar with the application icon and window controls. The main content area is light beige and contains three text input fields labeled "Connection String", "User Name", and "Password". At the bottom of the dialog, there are two buttons: "Connect" and "Cancel".

The user can then connect to the database by specifying the username and the password. Once the user connects to the database he sees the list of all the accounts which are open in the database. The user (usually the Administrator) can then set the various security privileges which he wishes to give to different users whose accounts are open. The following figure shows the screen after the user / Administrator logs in to the database.

After Logging to database – Before setting Security levels



After setting Security levels



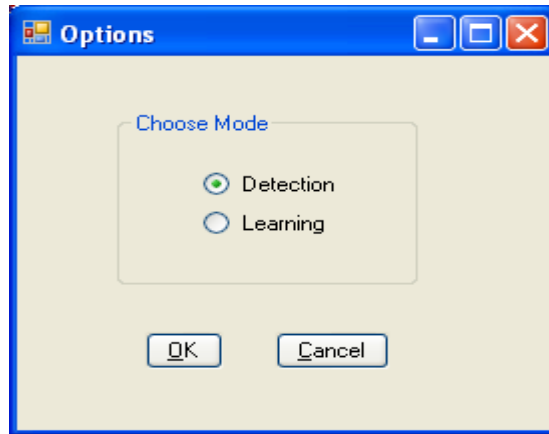
The user list can be displayed by selecting the list of users with an open account from the database and then displaying it on the MainScreen, the code for which is given below

```

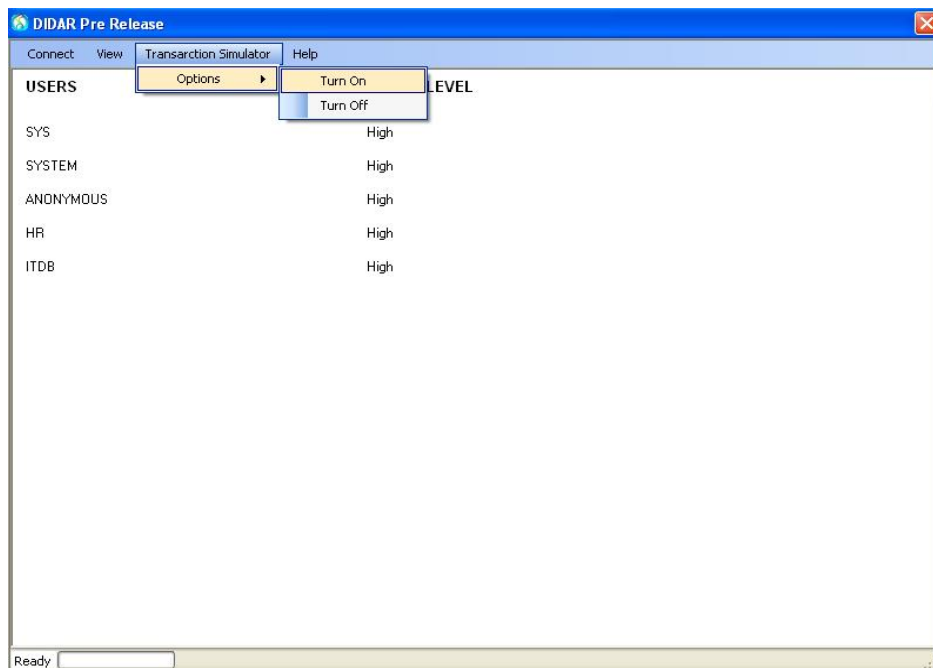
connstr=textBoxd.Text;
string ConnectForm.connstr=connstr;
pass = textBoxp.Text;
if (connstr.Equals(""))
{
    connstr = "Data Source=(DESCRIPTION=(ADDRESS_LIST="
+ "(ADDRESS=(PROTOCOL=TCP) (HOST=127.0.0.1) (PORT=1521)))"
+ "(CONNECT_DATA=(SERVER=DEDICATED) (SERVICE_NAME=XE)))";
}
MainScreen.ora = connstr + "User Id=" + user + ";Password=" + pass + ";";
OracleConnection conn = new OracleConnection();
conn.ConnectionString = MainScreen.ora;
conn.Open();
string sql = "select username from dba_users where account_status = 'OPEN'";
OracleCommand cmd = new OracleCommand(sql, conn);
cmd.CommandType = CommandType.Text;
OracleDataReader dr = cmd.ExecuteReader(); // C#
while (dr.Read())
{
    user_name[i] = dr.GetString(0);
    ITDB.Program.startScreen.arrLabels[i].Visible = true;
    //ITDB.Program.startScreen.arrButtons[i].Visible = true;
    ITDB.Program.startScreen.arrcomboBox[i].Visible = true;
    ITDB.Program.startScreen.arrcomboBox[i].Items.Add("Low");
    ITDB.Program.startScreen.arrcomboBox[i].Items.Add("Medium");
    ITDB.Program.startScreen.arrcomboBox[i].Items.Add("High");
    ITDB.Program.startScreen.arrcomboBox[i].Items.Add("Paranoid");
    ITDB.Program.startScreen.arrLabels[i].Text = user_name[i];
    //MessageBox.Show(sec_level[i]);
    i++;
    no_of_users++;
}
ITDB.Program.startScreen.label1.Visible = true;
ITDB.Program.startScreen.button1.Visible = true;
conn.Close();
this.Hide();
}
}

```

The user can then select either the option of Learning or Detecting intrusions for all the users. This is done by selecting the Options submenu in the Options form which opens up the options form as shown in the following figure. Once the user selects an option from the OptionsForm and clicks OK Button, the mode (Learning or Detection) is set to the choice of the user.



In order to simulate real querying conditions, we have also come up with a transaction simulator that can be used to simulate random fingerprints of queries. The transaction simulator can be turned On / Off. When the transaction simulator is turned on, the queries are generated randomly by the simulator and the action is done based on the choice of mode (Learning or Detection by the user). However it is best to ensure that the application learns completely before it starts to detect intrusions in to the database. When the transaction simulator is off, the queries are taken as the currently executing ones from the database.



Once the user decides on whether the transaction simulator is going to be switched On/Off then the real action begins. Then depending on the choice either the learning or detection algorithm is activated. If the Transaction Simulator is turned on then there are random queries generated which can either be not allowed in the user access graph or can be fingerprinted and then stored as an user access graph in a file. In detection, when a query is executing for a user, its fingerprint is checked with the user access graph for that particular user. If the fingerprint exists in the user access graph, then the transaction is allowed to execute. If it doesn't exist in the graph, then an appropriate action is taken according to the security level provided to the user. For example, if a malicious transaction is trying to execute for a user whose security is "Paranoid", then the transaction is blocked and written into a file of blocked transactions for that particular user to make future detections easier and less time consuming. The Learning and Detection for a simulated environment is given below.

```

for (int k = 0; k < ConnectForm.no_of_users; k++)
{
    for (int i = 1; i < max; i++)
    {
        cmd_type = random.Next(0, 3);
        no_of_rel = random.Next(1, 6);
        no_of_attr = random.Next(1, 9);
        no_of_cond = random.Next(1, 5);

        fingerprint_lts[i] = cmd_type.ToString() + "," +
            no_of_rel.ToString() + "," +
no_of_attr.ToString() +
no_of_cond.ToString();
        if (MessageBox.Show("Is this fingerprint
allowed?" +
"Learning", MessageBoxButtons.YesNo,
MessageBoxIcon.Question)== DialogResult.Yes)
        {
            fingerprintgraph[i][0] = fingerprintgraph[0][i] =
            fingerprintgraph[i][i] = "1";
            if (i > 1)
            {
                fingerprintgraph[i - 1][i] = "1";
            }
        }
    }
}

```

```

        }
    }
    else
    {
        i = i - 1;
    }
}
StreamWriter fwriter =
File.CreateText(ConnectForm.user_name[k]
+"_log.txt");

for (int i = 0; i < 20; i++)
{
    for (int j = 0; j < 20; j++)
    {
        if (j < 19)
        {
            fwriter.Write(fingergraph[i][j] + " ");
        }
        else
        {
            fwriter.Write(fingergraph[i][j]);
        }
    }
    fwriter.WriteLine();
}
fwriter.Close();
}

```

Similarly transactions executed in the database can also be fingerprinted. Any executing transaction from the database can be easily obtained along with the username and the type of the command executed by executing the query:

```
"select command_type,sql_text,username from v$sqlsession,v$sql
where sql_address = address and users_executing>0"
```

The above query returns the username of the user executing the query along with the query being executed. Thus by repeatedly executing the above query after some interval we can get the queries that are getting executed. The following is the method of fingerprinting a query executed by the database

```
public void learn(String temp, String user, String sec, int ind)
```

```

{
    if (selected_mode == "Learning")
    {
        fingerprint_l[ind] = temp;
        if (MessageBox.Show("Is this fingerprint allowed?" +
            fingerprint_l[ind], "Learning",
            MessageBoxButtons.YesNo, MessageBoxIcon.Question) ==
            DialogResult.Yes)
        {
            fingergraph[ind][0] = fingerprint_l[ind];
            fingergraph[0][ind] = fingerprint_l[ind];
            fingergraph[ind][ind] = "1";
            if (ind > 1)
            {
                fingergraph[ind - 1][ind] = "1";
            }
        }
        else
        {
            ind = ind - 1;
        }
        StreamWriter fwriter = File.CreateText(user_current +
            "_log.txt");
        for (int k = 0; k < 20; k++)
        {
            for (int j = 0; j < 20; j++)
            {
                if (j < 19)
                {
                    fwriter.Write(fingergraph[k][j] + " ");
                }
                else
                {
                    fwriter.Write(fingergraph[k][j]);
                }
            }
            fwriter.WriteLine();
        }
        fwriter.Close();
    }
}

```

However the only code difference between the two learning is that in the case of query executing from database the Learning function is called from a timer after fixed intervals so as to periodically get the list of executing transactions.

During detection the transactions coming in either from the transaction simulator or the database itself are first fingerprinted and then checked with the user access graph as stated already, and if not present, action is taken according to the designated security level of that particular user.

The code for the detection phase is shown below

Detection Phase:

```
String filename = user_current + "_log.txt";
StreamReader freader = File.OpenText(filename);
String text_line;
const char Space = ' ';
int count, code;
String[] output = new String[400];
while ((text_line = freader.ReadLine()) != null)
{
    count = 0;
    code = 1;
    char[] de_limiters = new char[]
    {
        Space
    };
    foreach (string subString in text_line.Split(de_limiters))
    {
        output[count] = subString;
        if (count < 20)
        {
            if (temp == output[count])
            {
                code = 1;
                MessageBox.Show("Fingerprint found in user
                access graph " + temp, "Found",
                MessageBoxButtons.OK,
                MessageBoxIcon.Information);

                break;
            }
            count++;
            code = 0;
        }
    }
    if (code == 0)
    {
        fpr = temp;
        int found_classify = 0;
        if (File.Exists(user_current + "_classifydata.txt"))
        {
            StreamReader str_read = File.OpenText(user_current +
            "_classifydata.txt");
            String strdt;
            while((strdt = str_read.ReadLine())!=null)
            {
                foreach(string subString in strdt.Split(' '))
                {
                    if(subString.Equals(fpr))
                    {
                        found_classify ++;
                    }
                }
            }
        }
    }
}
```

```

    }
    }
    str_read.Close();
}
if (found_classify >0)
{
    MessageBox.Show("Fingerprint execution Blocked");
}
else if (found_classify == 0)
{
    if (MessageBox.Show("Would you like to classify
        fingerprint as legal?", "Add Fingerprint?",
        MessageBoxButtons.YesNo,
        MessageBoxIcon.Question) == DialogResult.Yes)
    {
        MessageBox.Show("Fingerprint added in the user
            access graph", "Success",
            MessageBoxButtons.OK,
            MessageBoxIcon.Information);
    }
    else
    {
        String file_name = user_current +
            "_classifydata.txt";
        String risk_cat;
        if (String.Compare(no_of_tables.ToString(),
            Settings.risk_low) > 0)
        {
            if (String.Compare(no_of_tables.ToString(),
                Settings.risk_medium) > 0)
            {
                risk_cat = "High";
            }
            else
            {
                risk_cat = "Medium";
            }
        }
        else
        {
            risk_cat = "Low";
        }
        if (File.Exists(file_name))
        {
            StreamReader sr =
                File.OpenText(file_name);
            StreamWriter sw;
            String strdata, datastr;
            int exit_code = 1;
            while ((strdata = sr.ReadLine()) != null)
            {
                foreach (string subString in
                    strdata.Split(' '))
                {
                    datastr = subString;
                    if (String.Equals(datastr,
                        fpr))

```

```

        {
            exit_code = 0;
            break;
        }
    }
}
sr.Close();
if (exit_code == 1)
{
    sw =
        File.AppendText(file_name);
    sw.Write(fpr + " " + risk_cat);
    sw.WriteLine();
    sw.Close();
}
}
else
{
    StreamWriter sw1 =
        File.CreateText(file_name);
    sw1.WriteLine(fpr + " " +
        risk_cat);
    sw1.Close();
}
if (String.Compare(sec, "Paranoid",
                    true) == 0)
{
    is_blocked[userindex] = 1;
    String file_nm = user_current +
        "ITDB_blocked.txt";
    if (!(File.Exists(file_nm)))
    {
        fwriter2 =
            File.CreateText(file_nm);
        fwriter2.Write(fpr);
        fwriter2.WriteLine();
        fwriter2.Close();
        MessageBox.Show("Fingerprint
            has been classified as
            intrusion", "Intrusion",
            MessageBoxButtons.OK,
            MessageBoxIcon.Information);
    }
    else
    {
        StreamReader fr2 =
            File.OpenText(user_current
                + "ITDB_blocked.txt");
        String fnd = "";
        int found = 0;
        while ((fnd = fr2.ReadLine())
            != null)
        {
            //MessageBox.Show(fpr);
            //MessageBox.Show(fnd);
            if ((String.Compare(fpr,
                fnd, true)) == 0)

```

```

        {
            found = 1;
            MessageBox.Show
                ("Fingerprint"+
                fpr + " found in
                the Blocked list
                for the user " +
                user_current);
            break;
        }
    }
    fr2.Close();
    if (found == 0)
    {
        fwriter2 =
            File.AppendText
                (user_current +
                "ITDB_blocked.txt");
        fwriter2.Write(fpr);
        fwriter2.WriteLine();
        fwriter2.Close();

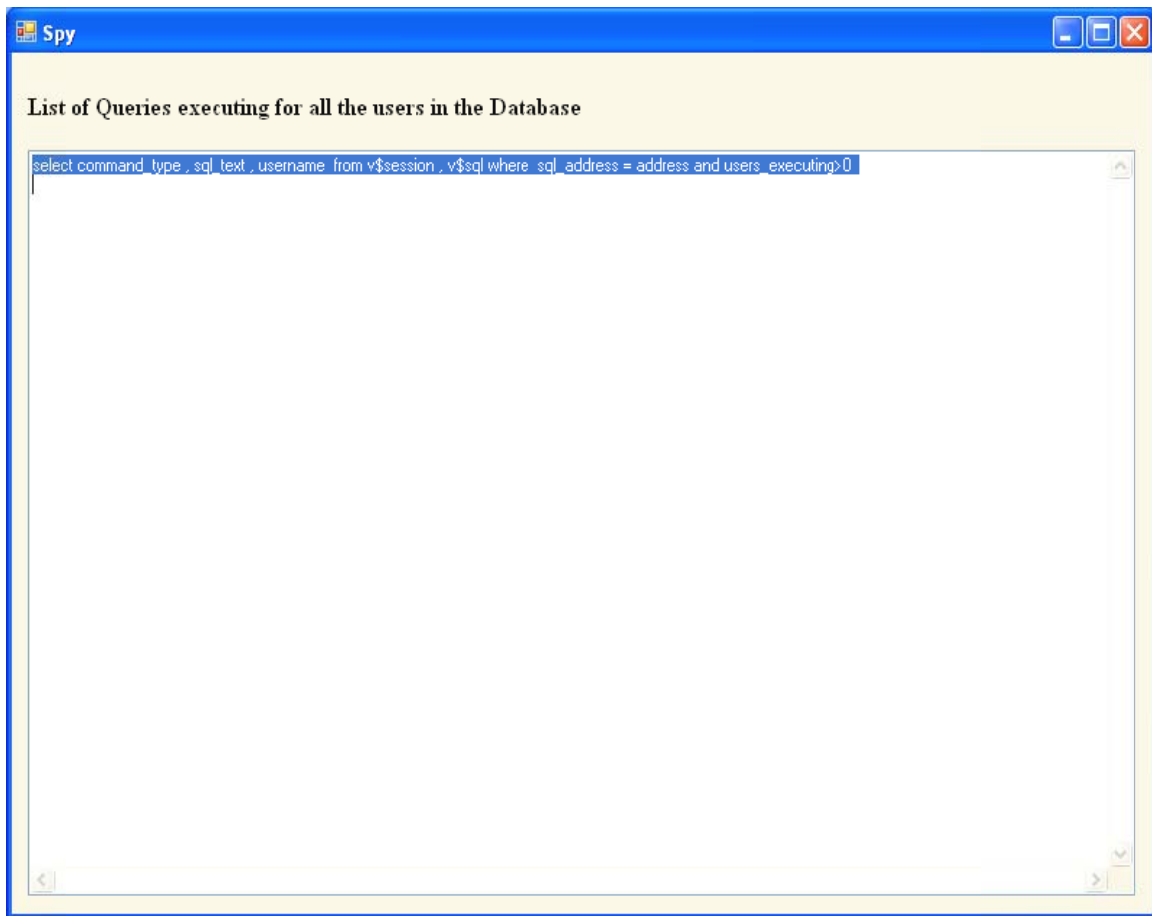
        MessageBox.Show
            ("Fingerprint has been
            classified as intrusion",
            "Intrusion",
            MessageBoxButtons.OK,
            MessageBoxIcon.Information);
    }

    }
    block_file++;
}
else if ((String.Compare(sec, "Medium",
true) == 0) ||
(String.Compare(sec, "High",
true)) == 0)
{
    lock_tables(tbl_names);
}
}
}
break;
}
}
}

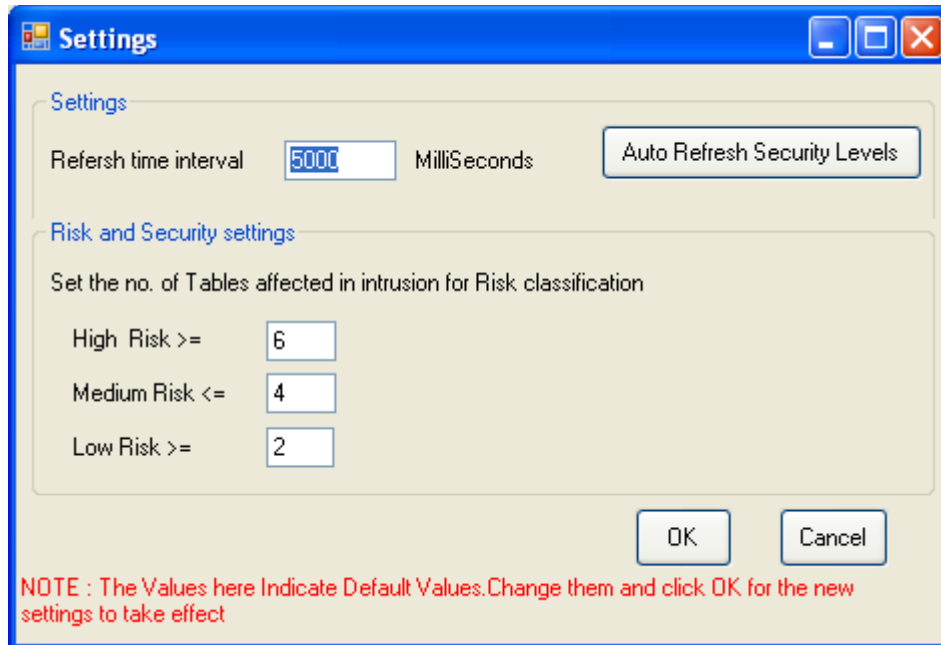
```

Again similar to learning here also the detection algorithm for queries executing on the database is called by a timer function after some regular interval of time

Also we allow the administrator to spy the database for queries running on it and allow him to change settings. When detecting, in addition to other approaches we also keep track of the intrusions in a file along with the risk factor associated. The risk factors are taken as High, medium and Low depending upon the damage that a transaction can do to the database. More the number of relations (Tables) it affects more will be the risk factor associated with it. The Settings option in the View Menu option allows the user to change the risk settings which are used for classification purposes. The following figures show the Spy form which displays all the currently executing transactions in the database.



The following figure shows the settings form where the user can view/modify the settings



The “**Auto Refresh Security Level**” button can be used to change the user security levels based on the classification of the risk factors mentioned already. The intrusions are prioritized for each user and accordingly the weighted mean is calculated as the threat posed by intrusions for each user. The **Percentage Threat** is then calculated and used to decide upon the security levels, the higher it is, more security the user needs. Percentage Threat is given by $(\sum R_i * P_i / N) * 100$.

Where,

R_i is the number of risks of level i .

P_i is the priority of risk level i .

N is the number of intrusions for a particular user.

6. Experimental Results

To actually verify the efficiency and performance measure of a system it is very important to develop relevant test cases to check whether the application is effectively meeting the requirements within the boundaries of the restrictions. Hence we came up with the idea of designing a transaction simulator for churning out random fingerprints of transactions to test our system. This section explores the test cases that were generated and the performance measures of our system in detecting malicious transactions that are intrusions

6.1 Test Cases

For a database system (Oracle 10g) with three users, namely **SYS**, **SYSTEM** and **ANONYMOUS** random fingerprints of transactions were generated and they were put to test on our learning and detection algorithms.

The fingerprints were generated randomly for each user and depending on the administrator's response of classification as an intrusion or a legitimate transaction's fingerprint; the user access graphs for each of the above mentioned user were created.

The fingerprinted transactions were then stored in a file as a matrix representing the user access graph. The graphs were then used for detection of similar random generation queries. The results and analysis of the detection and learning phase are discussed in detail in the following sub section.

Apart from using the fingerprints generated by the transaction simulator normal queries from the database were taken while executing and tested for detection. The discussion of the results of all these tests is in the following sub section

6.2 Test Results

After using the transaction simulator and a few of the queries that were executing in the database we found that the system was able to classify the intrusions with very good efficiency, when the learning is quite comprehensive. The efficiency was also aided largely due to the feedback mechanisms which helped in identifying intrusions and also legitimate transactions that were left out initially in the learning phase. Based on the administrator's response the following fingerprints were classified as legal and added to the user access graph of the respective users.

SYSTEM: <2,1,7,1> <3,2,3,4> <1,5,1,3> <0,3,1,1> <0,3,5,2> <1,2,5,1>
<1,3,7,2> <0,3,2,2> <1,4,7,3> <1,4,5,2> <1,1,8,3> <1,5,6,3>
<2,2,5,1> <2,2,3,3> <1,3,5,1> <1,2,7,1> <2,4,8,2>
<1,3,4,3> <0,2,1,2>

SYS: <0,2,8,4> <1,2,2,1> <0,3,6,2> <0,3,2,4> <1,5,5,4> <1,3,1,2>
<0,2,1,3> <1,1,7,2> <0,2,2,4> <1,3,1,1> <0,2,4,4> <1,2,4,1>
<2,1,3,3> <1,2,7,3> <0,4,7,1> <0,4,7,4> <2,1,2,4> <1,2,4,1>
<2,3,5,1>

ANONYMOUS: <0,4,3,4> <0,4,2,4> <2,3,3,4> <0,4,1,2> <1,5,1,1> <2,5,8,4>
<0,3,8,3> <2,5,5,1> <1,3,5,2> <0,3,1,2> <2,4,5,1> <2,2,5,2>
<2,3,7,3> <1,2,1,2> <2,3,7,3> <2,3,1,4> <2,2,3,3> <0,4,2,1>
<2,3,4,2>

Also it is important to note that the initial security level for the above users were set as **High for SYS, Medium for ANONYMOUS and Paranoid for SYSTEM**

During detection with the help of the feedback system the administrator was able to classify the intrusions along with classification of the legitimate ones left out during learning. The following were the contents of the files which contained the list of blocked transactions for each user.

SYS: (SYS_Blocked.txt): <0,5,5,3> <0,1,5,2> <1,3,6,3> <1,5,1,1> <2,1,4,2>< 0,1,4,4>

SYSTEM: (SYSTEM_Blocked.txt): <3,1,1,0> <3,1,1,3>

ANONYMOUS: (ANONYMOUS_Blocked.txt): <0,2,2,3>

Thus any further attempt to execute any of the blocked transaction is not permitted as these fingerprints will be maintained over time. In addition to this these and a few other transactions that were not a part of the blocking phase were used in classifying risk categories and threats of intrusions for various users. After the classification the users' security levels were successfully adjusted and finally all users were allotted a security level of **“Medium”**

In addition queries that were executing in the database were taken directly and fingerprinted and it was found that the results were same. Some queries like

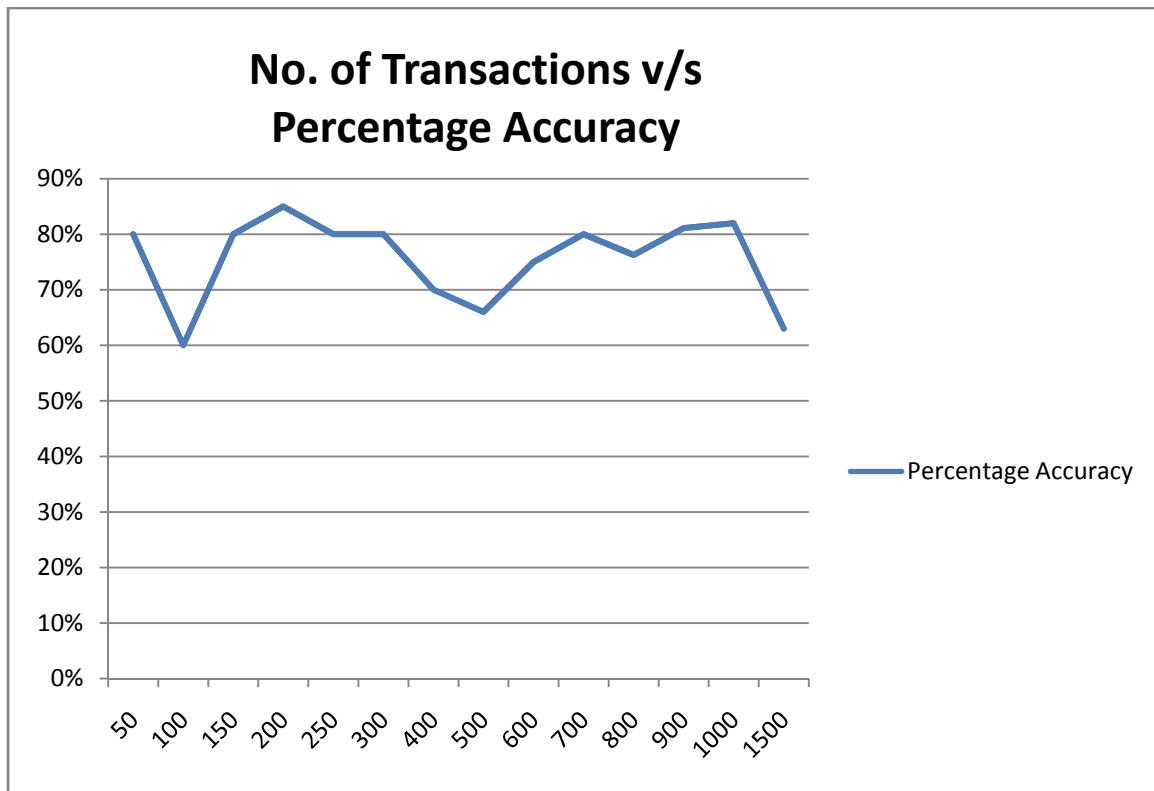
“Select * from user_tables;”

“Select username, password from v\$session”;

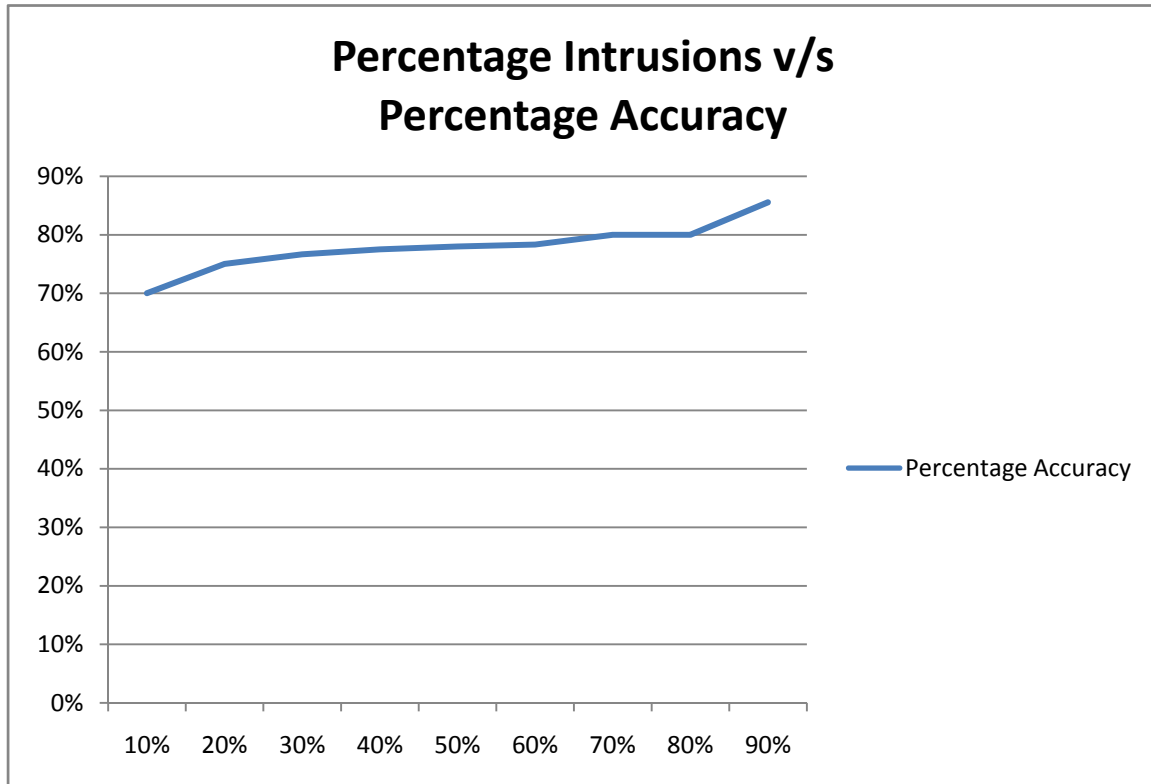
were executed over different security levels for the same set of users and was found that the timer if set at a reasonable interval can indeed detect all the queries that execute without any query being left out.

6.3 Experiments

We conducted several experiments using the transaction simulator. The first experiment consisted of observing the effect of increase in the number of transaction to the accuracy of the system. Keeping the percentage of intrusions constant we increased the number of transactions and get the following results.



In the other experiment we keep the number of transactions constant and keep on increasing the percentage of intrusions in the system. We get the following results.



It is clear from the above results that the percentage accuracy does not change with the increase in the number of transactions or the increase in the number of intrusions. The accuracy of the system remains constant between 60%-80% most of the time.

7. Conclusions and Future Work

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 a considerably low overhead to the database by itself.

For future work, the system can be tested on a live application say at a major bank or some other organization database. Another direction for future research is to maintain sub-graphs within a user access graph to capture the normal user behavior in a more intuitive sense. For example, consider the behavior of a reservation agent that needs to add bookings, modify bookings, cancel bookings, forward bookings, run statistics on bookings, etc. Each of these can be a separate class of behavior within the profile of a reservation agent role. The intrusion detection task can then be carried out as a combination of supervised and anomaly detection approaches.

References

- [1] Pramote Luenam, Peng Liu, The Design of an Adaptive Intrusion Tolerant Database System, Proceedings of the Foundations of Intrusion Tolerant Systems, 2003.
- [2] Peng Liu, Architectures of Intrusion Tolerant Database Systems, Proceedings of 18th Annual Computer Security Applications Conference, 2002.
- [3] Yi Hu, Brajendra Panda, A Data Mining Approach for Database Intrusion Detection, Proceedings of ACM Symposium on Applied Computing, 2004.
- [4] Abhinav Srivastava, Shamik Sural, A.K. Majumdar, Database Intrusion Detection using Weighted Sequence Mining, Journal of Computers, vol. 1, no. 4, July, 2006.
- [5] Wai Lup LOW, Joseph LEE, Peter TEOH, DIDAFIT detecting intrusions in databases through fingerprinting transactions, Proceedings of International Conference on Enterprise Information Systems, 2002.
- [6] Bertino, E. Terzi, E. Kamra, A. Vakali, Intrusion Detection in RBAC-administered Databases, Proceedings of 21st Annual Computer Security Applications Conference, 2005.
- [7] E. Lundin, E. Jonsson, Survey of Intrusion Detection Research, Technical Report Chalmers University of Technology, (2002).
- [8] W. Lee, S.J. Stolfo, Data Mining Approaches for Intrusion Detection, Proceedings of the USENIX Security Symposium, pp. 79-94 (1998).
- [9] D. Barbara, J. Couto, S. Jajodia, N. Wu, ADAM: A Testbed for Exploring the Use of Data Mining in Intrusion Detection, ACM SIGMOD, pp. 15-24 (2001).

- [10] C. Y. Chung, M. Gertz, K. Levitt, DEMIDS: A Misuse Detection System for Database Systems, IFIP TC-11 WG 11.5 Working Conference on Integrity and Internal Control in Information System, pp. 159-178 (1999).
- [11] V.C.S. Lee, J.A. Stankovic, S.H. Son, Intrusion Detection in Real-time Database Systems Via Time Signatures, Real Time Technology and Application Symposium, pp. 124 (2000).
- [12] Y. Zhong, X. Qin, Research on Algorithm of User Query Frequent Item sets Mining, Machine Learning Cybernetics, pp. 1671-1676 (2004).

Appendix

List of Diagrams

- a)** User access graph showing fingerprints.
- b)** Feedback loop with new node added
- c)** Information flow model of DIDAR
- d)** Architecture Learning Phase
- e)** Architecture Detection Phase
- f)** Architecture Data Warehousing Phase
- g)** Architecture Data Mining Phase
- h)** Class Diagram
- i)** DIDAR main screen screenshot
- j)** Connecting to database
- k)** Settings security level
- l)** Options menu
- m)** Mode selection learning or detection
- n)** Database spy tool
- o)** Risk and security settings

Keywords

Transaction – The smallest unit of the query executing in a database.

Intrusion – Malicious transaction which can cause damage to the database or affect the data consistency.

Fingerprint – The unique identifier for each transaction of the database. To create a fingerprint we look into each part of the select ... from ... where ... statement.

Security Levels – The levels in the application which determine the quality of information assurance provided by the framework. The higher the level more secure is the database. Choose from low, medium, high and paranoid.

Risk Levels -- Based on the content of the tables and the spread of the damage each intrusion can be classified with a risk level. There are three levels associated with risk – low, medium and high. Risk levels are used in the data mining phase to auto adjust the security levels.