Benchmarking Insider Threat Intrusion Detection Systems

by

Binbin Ye

A Thesis Submitted to Saint Mary's University, Halifax, Nova Scotia,
in Partial Fulfillment of the Requirements for the
Degree of Master of Science in Applied Science

March 25, 2013, Halifax, Nova Scotia

Copyright Binbin Ye, 2013

Approved:	Dr. Dawn Jutla Co-Supervisor Department of Finance, Computing Information Systems and Management Science
Approved:	Dr. Hai Wang Co-Supervisor Department of Finance, Computing Information Systems and Management Science
Approved:	Dr. Nur Zincir-Heywood External Examiner Department of Computer Sciences Dalhousie University
Approved:	Dr. Cristian Suteanu Supervisory Committee Member Department of Geography and Environmental Science
Approved:	Dr. Jeremy Lundholm Graduate Studies Representative
Date:	March 25, 2013

Abstract

Benchmarking Insider Threat Intrusion Detection Systems

by Binbin Ye

Abstract: An intrusion detection system generally detects unwanted manipulations to computer systems. In recent years, this technology has been used to protect personal information after it has been collected by an organization. Selecting an appropriate IDS is an important decision for system security administrators, to keep authorized employees from abusing their access to the system to exploit sensitive information. To date, little work has been done to create a benchmark for small and mid-size organizations to measure and compare the capability of different insider threat IDSs which are based on user profiling. It motivates us to create a benchmark which enables organizations to compare these different IDSs. The benchmark is used to produce useful comparisons of the accuracy and overhead of two key research implementations of future insider threat intrusion algorithms, which are based on user behavior.

March 25, 2013

TABLE OF CONTENTS

Abstract..**i**

Table of Contents ....**ii**

List of Table ....**iv**

List of Figures..**v**

List of Abbreviations..**vii**

1. Introduction..**1**

1.1 Overview of existing problems..1

1.2 Objective and goals...2

2. Literature Review **3**

2.1 Intrusion Detection in Database Systems 3

2.1.1 Data Dependency Relationship Based IDDS..............................4

2.1.2 Time Signature Based IDDS...5

2.1.3 User Behavior Profile Based IDDS 6

2.2 Benchmarking IDS...10

3. Benchmark Design...**15**

3.1 Selection of Performance Metrics...15

3.2 Selection of Parameters...22

3.3 Workload Design...24

3.3.1 Financial Bank Scenario 25

3.3.2 Transaction Template 30

3.3.3 Daily Transactions Scheduler 49

3.3.4 Types of Insider Attacks...53

4. Implementation **56**

4.1 Benchmark Implementation..56

4.1.1 Database Populator Implementation...57

4.1.2 Employee Profile Generator Implementation..............................61

4.1.3 Anomaly Generator Implementation 62

4.1.4 Anomaly Injector Implementation 64
4.1.5 Workload Driver Implementation 64
4.2 Implementation of Two Testing IDS 65
4.2.1 Implementation of RBAC Classifier 65
4.2.2 Implementation of Bayesian Network IDS 68
5. Experiment Methodology and Results 72
5.1 Scalability Testing 72
5.2 Different Anomaly Injection Methods Testing 80
5.3 Tuning IDS Settings with Benchmark 83
6. Conclusion and Future Work 86
6.1 Conclusion 86
6.2 Future Work 87

REFERENCES 88

List of Tables

Table 1 Conditional Probabilities of the Detector’s Report Given the State of the System 18
Table 2 Bank Database Schema 28
Table 3 Business Functions and Their Roles 31
Table 4 Daily Transactions Scheduler Fields 50
Table 5 Trans Table’s Attributes Description 65
Table 6 Triplet Construction Example 68

List of Figures

Figure 2.1 Overview of the RBAC-Specific ID Process 7
Figure 2.2 Entity Relationship Diagram for Bank Enterprise...................................12
Figure 3.1 Synthetic Benchmark Data; False Alarms vs. Regularity Index 20
Figure 3.2 Financial Bank Entity-Relationship Diagram 26
Figure 3.3 Table's Relationships in a Bank Database Implemented in MS SQL Server 29
Figure 3.4 A HR Employee Transaction Log for a Day Example 52
Figure 4.1 Benchmark Workload Flow..56
Figure 4.2 A JDBC Driver Example Used in Database Populator.............................58
Figure 4.3 Class Diagram of Database Populator..59
Figure 4.4 An Example of Generated Employee Table 61
Figure 4.5 Employee Profile Generator Workflow..62
Figure 4.6 Anomaly Generator Members View..63
Figure 4.7 Java Codes for Masquerade() Function..63
Figure 4.8 Class Diagram of RBAC Classifier..69
Figure 4.9 BNIDS Diagram in Genie 70
Figure 5.1 Test Case 1: Precision vs. Number of Injected Anomalies.......................73
Figure 5.2 Test Case 1: Recall vs. Number of Injected Anomalies 75
Figure 5.3 Test Case 1: F-measure vs. Number of Anomalies..................................76
Figure 5.4 Test Case 1: Time Cost vs. Number of Testing Tuples.............................76
Figure 5.5 Test Case 2: Precision vs. Number of Tuples...78
Figure 5.6 Test Case 2: Recall vs. Number of Tuples...78
Figure 5.7 Test Case 3: Precision vs. Number of Roles...79
Figure 5.8 Test Case 3: Recall vs. Number of Roles 80
Figure 5.9 Test Case 4: Precision vs. Number of Anomalies with RBAC Classifier..81
Figure 5.10 Test Case 4: Recall vs. Number of Anomalies with RBAC Classifier...81

Figure 5.11 Test Case 4: Precision vs. Number of Anomalies with BNIDS..............82
Figure 5.12 Test Case 4: Recall vs. Number of Anomalies with BNIDS 82
Figure 5.13 BNIDS Threshold Tuning...83
Figure 5.14 Precision vs. Number of Injected Anomalies 84
Figure 5.15 Recall vs. Number of Injected Anomalies..85

List of Abbreviations

API	Application program interface
AUC	Area under the ROC curve
BNIDS	Bayesian Network Intrusion Detection System
CERT	Computer Emergency Readiness Team
CSI	Crime Scene Investigation
DARPA	Defense Advanced Research Projects Agency
DB	Database
DBIR	Data Breach Investigations Report
DBMS	Database management system
DEMIDS	Detection Misuse in Database Systems
ER	Entity Relationship
FN	False Negative
FP	False Positive
GeNIe	General Electric Network for Information Exchange
ID	Intrusion detection
IDDS	Intrusion detection in database system
IDS	Intrusion detection system
JDBC	Java Database Connectivity
jSMILE	Java Structural Modeling, Inference, and Learning Engine
LARIAT	Lincoln Adaptable Real-time Information Assurance Test-bed
MAP	Maximum Aposteriori Probability
NPV	Negative predictive value
OLTP	Online Transaction Processing
PIDS	Privacy Intrusion Detection System
PPV	Positive predictive value
QID	Query Intrusion Detector

RBAC	Role-based access control
ROC	Receiver Operating Characteristic
SQL	Structured Query Language
TN	True Negative
TP	True Positive
TPC-A	Transaction Performance Council – A benchmark
TPC-B	Transaction Performance Council – B benchmark
TPoX	Transaction Processing over XML
URL	Uniform Resource Locator

Chapter 1

Introduction

1.1 Overview of existing problems

An insider threat is a current or former member of an organization who has or had privileged access to classified, sensitive or propriety data and exceeded or misused that access in a way that puts the organization's confidentiality, integrity, or availability of the organization's information or information systems in jeopardy (CERT, 2010). In recent years, several computer crime surveys show that insider attack has become a major threat in cyber security. According to the 2011 CSI Computer Crime survey and the 2011 Verizon Data Breach Report data breach due to insider attacks cost more than those due to outsiders. Insider threat has becoming one of the problems of organizational security that is most difficult to handle because insiders often have information and capabilities not known to external attackers, and as a consequence can cause serious harm (Hunker & W.Probst, 2011). The biggest concern is the discrimination between legal insider actions representing a threat, and legal insider actions representing normal behavior. This is where many of the standard techniques fail, since they require a clear separation between insiders and outsiders, between "good" employees and attackers (Moore, Cappelli, Shaw, Spooner & Trzeciak, 2011).

In order to address this issue, efforts have been made to identify and prevent insider threats by using Intrusion Detection System (IDS) technology. An intrusion detection system generally detects unwanted manipulations to computer systems. In recent years, this technology has been used to protect personal information against insider threat after it has been collected and stored by an organization.

The performance of an Intrusion Detection System plays an integral role in the decision of a company to utilise that system to protect personal data against insider threat. Such decisions must be based on information that compares the performance and price of different Intrusion Detection Systems. Benchmarks provide a yardstick with which to measure these important factors. However, to date little work has been done to create a benchmark for small and mid-size organizations to measure and compare the capability of different insider threat IDS which are based on user profiling. It motivates us to create a benchmark standard which enables larger organizations to compare these different IDS. The benchmark is used to produce useful comparisons of the accuracy and overhead of two key research implementations of future privacy intrusion algorithms that are based on user behavior in the context of a large organization.

1.2 Objectives and Goals

The goals of this paper are to design and implement a benchmark for insider threat intrusion detection systems which are based on user profiling, in the context of small to mid-size organizations, and to demonstrate the usefulness of the benchmark through its application to measure and compare different insider threat intrusion detection algorithms based on user behavior.

Chapter 2

Literature Review

In this chapter, we first introduce three popular intrusion detection techniques that have been used for protecting information stored in database systems – data dependency relationship based IDDS, time-signature based IDDS and User behavior profile based IDDS. In the second part of this chapter, the benchmarking procedures that have been used in database systems are discussed.

2.1 Intrusion Detection in Database Systems (IDDS)

The problem of developing Database Management Systems (DBMS) with confidentiality guarantees and high-assurance privacy is not that trivial (Agrawal, Kiernan, Srikant & Xu, 2002). To start with, it requires a revision of architectures and techniques adopted by current DBMS.

Despite the necessity of protecting information stored in database systems (DBS), existing security models are insufficient to prevent misuse, especially insider abuse by legitimate users. Further, concepts for misuse detection in DBS have not been adequately addressed by existing research in misuse detection. Even though there are some means to guard the information stored in a database system against misuse, they are seldom used by security officers because security policies of the organization are either imprecise or not known at all (Antón, Bertino, Li & Yu, 2004). Thus, intrusion detection systems have emerged as independent and complementary software for additionally securing DBSs. Examining the state-of-the-art, we can divide intrusion detection techniques into three main types: data dependency relationship-based; time-based; and user behavior profiles-based Intrusion Detection in Database Systems (IDDS). In the following sections, we will discuss the works that have been done with these three types of IDS techniques in detail.

2.1.1 Data Dependency Relationship Based IDDS

In data dependency relationship-based IDDS, a system tries to find malicious database transactions submitted to the DBMS by analyzing the dependencies among the data items in the database. In recent research, data dependency refers to the data access correlations between two or more data items (Hu & Panda, 2003). That is, which data items must be read or written before a data item gets updated and which others are written after the update. Those data items are separately known as read set, pre-write set and post-write set. They focused on the malicious transactions issued by an intruder masqueraded as a normal user. By checking whether each update operation in the user transaction conforms to the generalized data dependencies, anomaly activities at the transaction level are detected.

A static semantic analyzer is used to analyze the database application program instead of the database log, to decide the read set, the pre-write set, and the post-write set. First the system finds out all the possible transactions one user may use, which can be identified by checking the database application program. Then the static semantic analyzer is used to check all statements that update data items in each transaction to find out the read, pre-write, and post-write sets for each data item that is updated in the transaction. Other statements that are not for updating purpose are not checked.

Hu and Pamda (2003) also propose the method for finding anomalies at the user task level, which are comprised of a group of transactions, by using a system modeling tool called Petri-Nets to model normal data update sequences in user tasks. This method is especially useful for finding hidden malicious activities that consists of several transactions, each of which appears as normal transaction.

2.1.2 Time Signature Based IDDS

Real-time database systems have to deal with data which changes its value with time. These temporal data objects are used to reflect the status of objects in the real world. Whenever the value of a real world object changes, the data value that describes this object should change as well, but a certain lag between the moment of change in the real world and the updates in the database is unavoidable (Bodlaender, Stock & Son, 1997). Therefore, data values may become out of date due to delayed updates. If intruders are able to change such values, they could sabotage applications such as air traffic control or the electric power grid.

Lee, Stankovic & Son (2000) monitor behavior at the level of sensor transactions, which they define as transactions that are responsible for updating the values of real-time data. By determining the update latency of sensor transactions in real-time database systems, violations of the security policy can be detected. In order to determine the period of the sensor transaction, they consider one of the many possible semantics of sensor transactions with period P. One instance of the sensor transaction must be generated every period. Suppose a sensor transaction takes e units of time to complete (0<e<P). If an instances starts at time t, they expect the corresponding temporal data will be updated between t+e and t+p. Security alarm will be triggered when a transaction attempts to write a temporal data object that is already updated in a period. In addition, they also allow different tolerance levels in a system by utilizing the knowledge of the behavioral constrained (sensitive) IDS.

2.1.3 User Behavior Profile Based IDDS

Behavior profiles based IDDS are more popular than data dependency relationship-based or time-based IDDS. Systems attempt to profile normal user behavior by analyzing the database log or database application program.

The approach used in (Bertino, Kamra, Terzi & Vakali, 2005) is based on mining database traces stored in log files. The result of the mining process is used to form user profiles that can model normal behavior and identify intruders. By considering the fact that it is not feasible in practice keeping a profile for each single user, Bertino et al, (2005) have based their approach on the well-known role-based access control (RBAC) model, which is widely used for access-control management both in closed and open systems (Forrest, Hofmeyr & Somayaji, 1997). Through building profiles for each role, their ID system is able to determine role intruders, that is, individuals that while holding a specific role, have a behavior different from the normal behavior of the role. Authorizations are specified with respect to roles and not with respect to individual users.

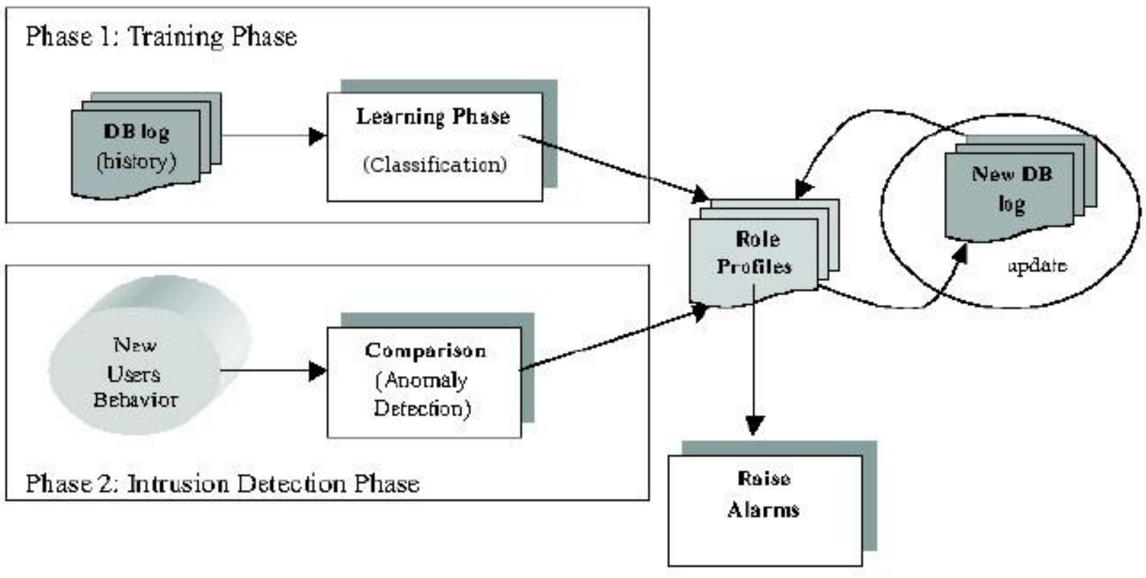


Figure 2.1 Overview of the RBAC-Specific ID Process (Bertino, et al, 2005)

As we can see in the Figure 2.1, the system's architecture consists of four main components: the user that enters queries, the conventional DBMS mechanism that handles the query evaluation process, the database log files and the ID mechanism. Every time a query is issued, the database log files are updated. In the training phase, the intrusion detection system mines the existing log files and forms role profiles. In the detection phase, for every new query, the ID mechanism checks the query statement to determine whether it is anomalous. If this is the case, an alarm is raised

(Agrawal et al, 2002).

In the paper, they transform the log file entries into a format that can be processed and analyzed. Therefore, they represent each entry by a data basic unit that contains three fields (SQL Command, Relation Information, Attribute Information), and thus it is called a triplet. A Naive Bayes Classifier is used for forming the profiles as well as for deciding when to raise an intrusion alarm. The Maximum Aposteriori probability (MAP) decision rule is used by the Classifier.

An, Jutla & Cercone (2006) provide a theoretically significant improvement to the intrusion detection algorithm proposed by Bertino et al, (2005). Instead of using the simple Naïve Bayes (Naïve BN) approach, a Bayesian Network method is used. The latter removes the Naïve Bayes assumptions that all classification features are independent of each other. The Bayes Network method allows the combination of user activities to be modeled together and improves the credibility of the alarms raised by the intrusion detection system. However the Bayes Network model introduces further overhead, over Naïve BN, due to the requirement for compiling the BN into a tree structure to perform inference and the manipulation of the resulting larger conditional probability distribution tables.

The idea of using an IDS approach to protect privacy is not new. User profile-based methods are presented in DEMIDS (Detection Misuse in Database Systems) (Chung, Gertz & Levitt, 1999), which is a misuse detection system tailored to relational database systems. It uses audit-log data to derive profiles that describe typical behavior (access patterns) of users working with the DBS by specifying the typical values of features that are audited in audit logs. Carter and Katz (2005) revealed that in computer systems the primary security threat comes from insider abuse rather than from intrusion. This observation results in the fact that much more emphasis has to be placed on internal control mechanisms of systems, such as audit log analysis. Thus in DEMIDS the profiles computed can be used to detect misuse behavior, in particular insider abuse. Furthermore, the profiles can serve as a valuable tool for security reengineering of an organization by helping the security officers to define/refine security policies and to verify existing security policies, if there are any.

Moreover, the approach used by DEMIDS is that the access patterns of users typically form some working scopes which comprise certain sets of attributes that are usually referenced together with some values in a query. The idea of working scopes is conceptually captured by the concept of frequent itemsets which are sets of features with certain values. Based on the data structure and semantics (integrity constraints) encoded in the data dictionary and the user behavior reflected in the audit log, DEMIDS defines a notion of distance measure which measures the closeness of a set of attributes with respect to the working scopes. Distance measures are used to guide the search for frequent itemsets in the audit logs by a novel data mining approach. Anomalies can be detected by comparing the derived profiles against the security policies specified or against new information (audit data) gathered about users.

Another behavior profiles-based IDDS is PIDS (Venter, Olivier & Eloff, 2004), which is an anomaly intrusion detection system. PIDS particularly addresses the problem of the internal misuses of private data within an organization, where profiles of the operators are represented by some features. Once the threshold regarding a feature is violated, the respective action will be taken to slow down or stop the possible anomaly activities. In PIDS, it also uses Naive Bayes to decide whether the employee's behavior is anomalous or not.

In a Hippocratic Database (Agrawal et al, 2002) a Query Intrusion Detector (QID) is proposed. PIDS differs from the QID in three significant respects: PIDS considers queries while QID considers the results of queries before data is released. Secondly, PIDS uses an intrusion detection model based on the expected activities of a user. This model is derived from the role of the user, as well as individual traits. In contrast QID builds a profile from past queries. Thirdly QID apparently only flags suspect queries, whiles PIDS attempts to limit damage by using throttling (Venter, et al, 2004).

2.2 Benchmarking Intrusion Detection Systems

By discussing the architecture and functionality of different kinds of Intrusion Detection in Database System above, we illustrate that there are reasons for expected differences in the system performance. Plus, organizations have a need for comparing and understanding these differences for their contexts. We focus on specifying a benchmark for the family of user behavior profile-based IDS. Choosing suitable performance measures and how to generate a workload to train, test and compare the

IDS are two main concerns for benchmarking.

A recent 2012 study shows that within all the industry groups being investigated for data breach incidents, Finance and Insurance responds to 40% of compromised records with more than 1 Million records lost across industries. In this report, it also indicates that Finance and Insurance ranks on top with 28% of breaches within larger organizations (DBIR, 2012). Hence, it makes sense to build our benchmark in a financial services scenario where insider threats are most likely to occur. The idea of using a financial bank scenario in a database benchmark is not new. The first transaction processing benchmarks, DebitCredit, TPC-A and TPC-B, are designed to measure throughput in terms of how many transactions are processed per second in a particular banking enterprise. The DebitCredit benchmark emulates a teller support system in a large, multi-branch bank, where the operation of interest is a debit or credit transaction to an account performed by an account holder at a particular branch (Dietrich, Brown, Cortes-Rello & Wunderlin, 1992). The DebitCredit database stores information on accounts, tellers, branches, and the history file of bank transactions. Figure 2.2 is the Entity-Relationship (ER) diagram for the bank enterprise used in the DebitCredit benchmark. TPC-A is an industry standard for DebitCredit benchmark founded by Transaction Processing Performance Council (TPC), followed by TPC-B a database only version of TPC-A benchmark. However, while both benchmarks are designed to measure the performance of the update-intensive operations in the database, neither of them includes insider attack scenarios.

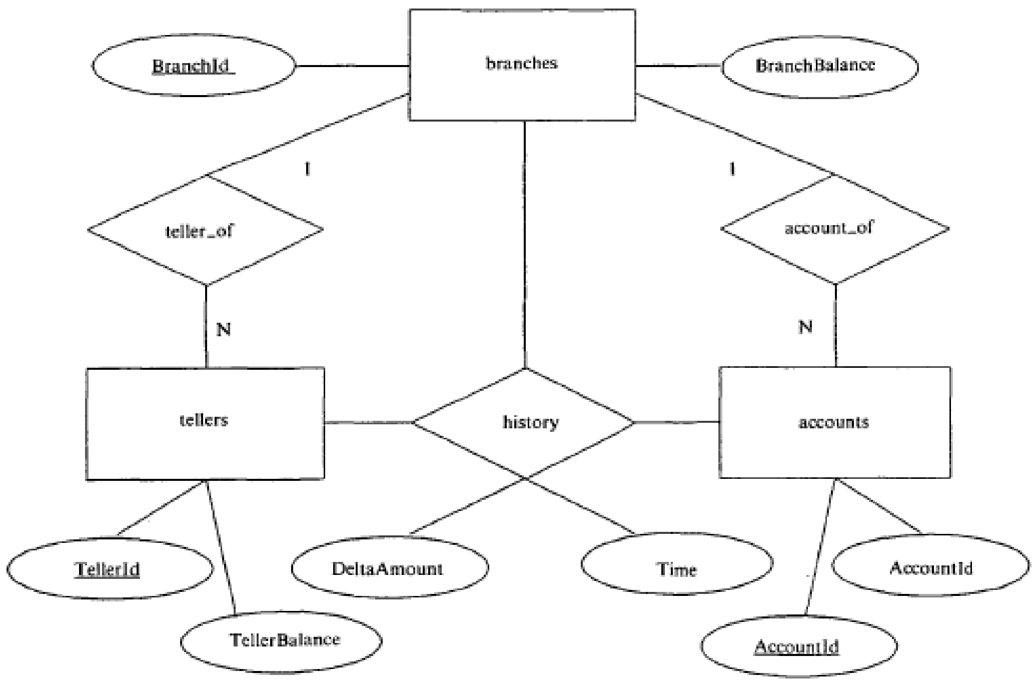


Figure 2.2 Entity Relationship Diagram for Bank Enterprise

Most if not all benchmarks for Intrusion Detection Systems are targeting outsider threats, which can be representing as a hacker who does not have an authorized access and breach into the systems. Most intrusion detection systems rely on some kind of pattern-matching algorithm in order to identify and categorize attacks. After training against "normal" data, the IDS can diagnose whether an action is an anomaly by matching it against normal patterns. The performance of an intrusion detection system is determined by the efficiency with which it diagnoses those attacks. Defense Advanced Research Projects Agency (DARPA) Intrusion Detection Evaluation is the first attempt to use simulated workload that involve generating background traffic interlaced with malicious activity, so that intrusion detection systems and algorithms could be tested and compared(Kayacik & Zincir-Heywood, 2005). Later on, the Lincoln Adaptable Real-time Information Assurance Test-bed (LARIAT) was initiated. Its effort is focusing on custom software that simulates network traffic from a small network connected to the Internet. The most common methodology of an IDS benchmark is to create a small test network and release malicious activity in that self-contained small network environment. The workload, i.e. network architecture and complexity of the traffic, is one of the main concerns for IDS benchmark design for online systems.

In benchmarking parlance, the training data (normal background) and testing data (back-ground plus anomalous events) constitute the anomaly-detector workload. The methods proposed by Maxion and Tan (2000) illustrate that the training data are generated from 11 transition matrices that produce the desired regularities for the sequences such that the regularity indices of the sequences run, in increments of .1, from 0 to 1 inclusive. The transition matrix is entered at a random point, determined by a random number generator. Test data were generated in the same way in which the training data were generated, except that different random-number-generator seeds were used for generating the test data. The seed used for generation is different from the one used to enter the table, simply to go as far as possible to eliminate dependencies in the generation scheme. A single seed is used to generate data for all regularities, the seed is a 4-digit random integer produced from the Perl rand( ) function. A pool of anomalies for each anomaly type is generated independent of generating the test data and then injected into the test data.

Anomaly detector works in two phases: a learning phase and a detection phase. Simply described, in the learning phase it constructs an internal table containing the probability of occurrence of every unique n-gram in the training sequence (e.g. normal background data). In the detection phase, it is given a test sequence consisting of normal background data (noise) mixed with injected anomalous-event subsequences (signal).

Scoring the detection outcomes: event outcomes in terms of hits (correct detection), misses (the absence of a detection within the basic scope of the injected anomaly) and false alarms (false detections any detection that falls outside the total scope of the injected anomaly).

In RBAC-specific ID (Bertino et al, 2005), similarly, the workload is comprised of Synthetic Data sets, Real Data set and Intruder Queries. The real data set used for evaluating the proposed consists of over 6000 SQL traces from eight different applications submitting queries to a MS SQL server database. The database itself consists of 119 tables with 1142 attributes in all. The intruder/anomalous queries are generated by taking into account the insider threat scenario. They are taken from the same probability distribution as of normal queries, but with role information negated. For example, if the role information associated with a normal query is 0, then simply change the role to any role other than 0 to make the query anomalous.

Chapter 3

Benchmark Design

In this chapter, we first identify relevant requirements and design decisions and then describe the workload design for the benchmark. The procedure proposed in (Jain, 1991) is being used as a reference model of how to proceed in benchmark design. The goal of this benchmark is to measure and compare different e-privacy intrusion detection algorithms based on user behavior. There is only one tested service: user behavior. For this service, a collection of tests is proposed, each of which focus on a specific sub-task of active behavior. The selection of goals and services should be fair, i.e., it should not favor specific systems. The benchmark tests the features that are common to most current relational database systems, and does not stress special features that are available for a few systems. The benchmark is designed to scale as the bank expands and new employees are hired, which can be reflected by incrementing branches, customers and employees.

3.1 Selection of metrics

A fundamental problem in intrusion detection benchmarking is to objectively evaluate an intrusion detection system (IDS) in terms of its ability to correctly classify events as normal or intrusive. The most basic and commonly used metrics are true positive rate (TP, i.e., the probability that the IDS outputs an alarm when there is an intrusion) and false positive rate (FP, i.e., the probability that the IDS outputs an alarm when no intrusion occurs). Alternatively, one can use a false negative rate FN=1-TP and true negative rate TN=1- FP (Gu, Fogla, Dagon, Kee & Skori, 2006).

However, by setting the threshold of a deviation from a normal profile, there may be different TP and FP values associated with different IDS operation points. If only the metrics of TP, FP is given individually, determining the better operation point is difficult. Thus, a composite metric, in which both TP and FP are considered, is necessary (Gu, et al, 2006).

Traditional standard composite quality metrics of IDS are precision and recall, which are defined as follows:

$$Precision = \frac{\text{\# True Positives}}{\text{\# True Positives} + \text{\# False Positives}}$$

$$Recall = \frac{\text{\# True Positives}}{\text{\# True Positives} + \text{\# False Negatives}}$$

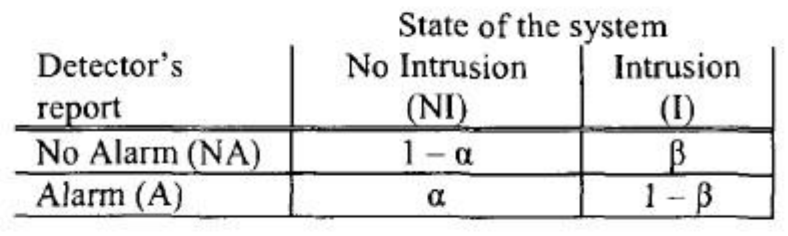
Here, # false Positives is the number of false alarms while # false Negatives is the number of times the system is not able to detect the anomalous queries (Agrawal; Kiernan; Srikant; & Xu, 2002). Precision is defined as the ratio of the number of correctly detected anomalous behaviors to the total estimated anomalous behaviors, and the recall is the hit-rate, that is, the ratio of the number of correctly detected anomalous behaviors to the total number of anomalous behaviors (Yao, An & Huang, 2005).

Another popular composite metric is the Receiver Operating Characteristic (ROC) curve, which has been introduced to evaluate machine learning algorithms, to plot the different TP and FP values associated with different IDS operation points (Zeng, 2005). It shows the probability of detection provided by the detector at a given false alarm rate. However, Gu et al, (2006) argue that ROC curves cannot be effectively used for comparing IDSs, as when the curves cross it is not easy to compare the IDSs. It is not always appropriate to use the area under the ROC curve (AUC) for comparison because it measures all possible operation points of an IDS. Calzarossa & Serazzi, (1993) also argue that evaluations based on ROC analyses are often misleading. They further argue and demonstrate how evaluations based on cost metrics can overcome some of the problems of ROC analysis.

One approach to integrating the metrics TP and FP is through cost-based analysis. Essentially, the tradeoff between a true positive and a false positive is considered in terms of cost measures (or estimates) of the damage due to an intrusion, and inconvenience caused by a false alarm. Gaffney and Ulvila (2001) used such an approach, which is similar to the approach proposed by Calzarossa & Serazzi, (1993), to combine ROC curves with cost analysis to compute an expected cost for each IDS operation point. The expected cost can be used to select the optimal operation point on a detector’s receiver operating characteristic (ROC) curve and to compare different IDSs. The detector’s ROC curve describes the relationship between the two operating parameters of the detector, its probability of detection, 1-β, and its false alarm rate, α. That is, the ROC curve displays the1-β provided by the detector at a given α. It also displays α provided by the detector at a given1-β. The quality of cost-based analysis depends on how well the cost estimates reflect the reality. However, Gu et al, (2006) mention that cost measures in security are often determined subjectively because of the lack of good (risk) analysis models. Thus, cost-based analysis alone cannot be used to objectively evaluate and compare IDSs.

Table 1

Conditional Probabilities of the Detector's Report Given the State of the System (Gue, et al, 2006)



	State of the system	
Detector's report	No Intrusion (NI)	Intrusion (I)
No Alarm (NA)	$1 - \alpha$	β
Alarm (A)	α	$1 - \beta$

In addition to TP and FP, two other useful metrics are the positive predictive value(PPV), which is the probability of an intrusion when the IDS outputs an alarm, and the negative predictive value (NPV), which is the probability of no intrusion when the IDS does not output an alarm. The Positive Predictive Value (PPV) is defined as follow:

$$P(I|A) = \frac{P(I,A)}{P(A)} = \frac{P(I)P(A|I)}{P(I)P(A|I) + P(\neg I)P(A|\neg I)}$$

Similarly, the Negative Predictive Value (NPV) is

$$P(\neg I|\neg A) = \frac{(1-B)(1-\alpha)}{(1-B)(1-\alpha)+B\beta}$$

In (Gu, Fogla, Dagon, Kee & Skori, 2006), it states that these metrics are very important from a usability point of view because ultimately, the IDS alarms are useful to an intrusion response system (or administrative staff) only if the IDS has high PPV and NPV. Both PPV and NPV depend on TP and FP, and are very sensitive to the base rate (B), which is the prior probability of intrusion. Thus, these two metrics can be expressed using Bayes theorem so that the base rate can be entered as a piece of prior information about the IDS operational environment in the Bayesian equations. Similar to the situation with TP and FP, both PPV and NPV are needed when evaluating an IDS from a usability point of view, and currently, there is no objective method to integrate both metrics.

Most anomaly detectors are based on probabilistic algorithms that exploit the intrinsic structure, or regularity, embedded in data logs. Maxion and Tan (2000) introduce a metric for characterizing structure in data environments, and tests the hypothesis that intrinsic structure influences probabilistic detection. They argue that most evaluations are done according to a black-box testing regime, which can demonstrate the overall performance capabilities of a detection system but reveals almost nothing about the performance of components inside the black box.

Because most, if not all, anomaly-detection algorithms depend on the intrinsic structure embedded in the data upon which they operate, it seems reasonable to assume that differences in such structure would influence detector performance. High-regularity data contain redundancies that facilitate predicting future events on the basis of past events; low-regularity data impede prediction. Regularities refer to the sequential dependencies of sequences as measured by entropy, which has been called the Shannon-Wiener Index also (Maxion & Tan, 2000).

Maxion et al, (2000) propose benchmarking as an approach toward understanding the performance characteristics of anomaly-detection algorithms applied to categorical data.

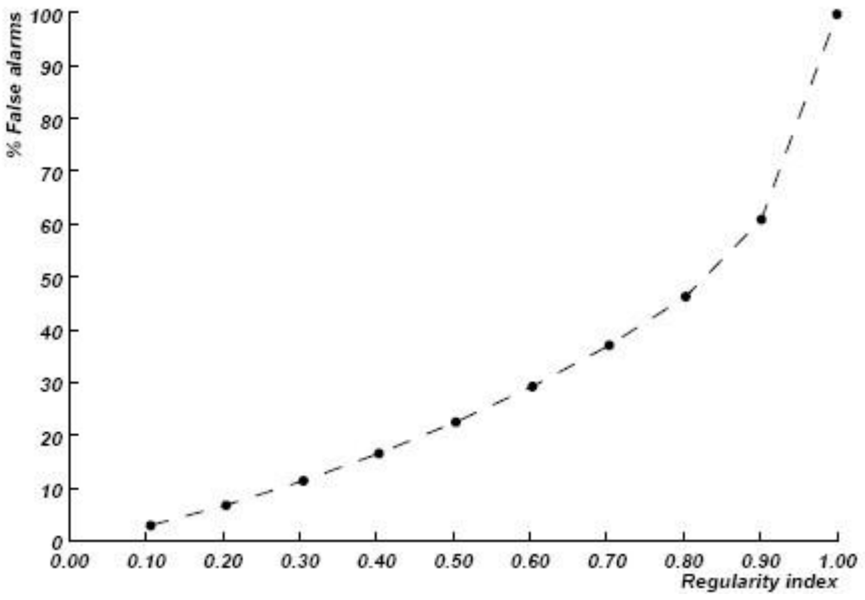


Figure 3.1 Synthetic Benchmark Data; False Alarms vs. Regularity Index (Maxion & Tan, 2000)

Another performance measure, the F-measure used to evaluate the accuracy of the session detection is presented by Yao, An & Huang, (2005). The F-measure is defined as a harmonic mean of precision (P) and recall (R) (van Rijsbergen, 1979).

The harmonic mean H is defined as follows.

$$H = \frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}} = \frac{n}{\frac{1}{x_1} + \cdots + \frac{1}{x_n}}$$

When x_1= P and x_2=R, H will be:

$$\textit{F-measure} = H = \frac{2}{\frac{1}{P} + \frac{1}{R}}$$

$$= \frac{2}{\frac{P+R}{PR}} = \frac{2PR}{P+R}$$

F-measure has been used in information retrieval to measure the retrieval performance (Lundin & Jonsson, 2000)(Huang & Peng, 2004). A higher F-measure value means a better overall performance. The definition of F-measure as mentioned above is also called F_1-measure by some researchers. The full definition of the F-measure is given as follows (Chinchor, 1992):

$$F_\beta = \frac{(\beta^2+1)PR}{\beta^2 P + R} \quad (0 \leq \beta \leq +\infty), \text{ where } \beta = \frac{R}{P} \ .$$

β is a parameter that controls a balance between P and R. When β =1, F_1 comes to be equivalent to the harmonic mean of P and R. If β >1, F becomes more recall-oriented and if β<1, it becomes more precision-oriented, e.g., F_0=P.

We list three popular composite performance metrics above - ROC curves, regularity index, and F-measure. In the future work of the benchmark design, we will include F_1-measure as one basic performance metric in addition to precision and recall. The reason we do not chose ROC curve in our benchmark is that it cannot be effectively used for comparing IDSs, and it is not always appropriate to use the area under the ROC curve (AUC) for comparison because it measures all possible operation points of an IDS (Calzarossa & Serazzi, 1993) (Gu et al, 2006). We do not include regularity index as well is because the benchmark focuses on transaction level instead of data level performance.

3.2 Selection of Parameters

Parameters are factors that influence IDSs performance on benchmark. By tuning configuration parameters based upon different benchmark testing needs, comparable results can be produced. Below we identify the specific parameters for e-privacy IDSs performance measurement.

records accessed The total number of records been accessed per transaction (Maxion & Tan, 2000)

tables accessed The total number of tables been accessed per transaction (Maxion & Tan, 2000)

transaction type The most common transaction types are listed as followed. *Check*: Check account information; *Deposit*: Deposit money into the account *Withdraw*: withdraw money from an account; *Transfer In*: Money moved into this account from another bank account, or money received from the bank holding this account; *Transfer Out*: Money moved out of this account into another bank account, or money paid to the bank holding this account.

database size Database size is determined by individually specifying the total number of branches, departments , customers and employees, and the number of records in each.

table size The number of records generated on individual tables

number of simulated users Total number of simulated users of the database. In our benchmark, it represents the number of employees executing transactions on the banking system.

user roles The total number of roles that exists in the benchmark workload. More details on role design will be provided in Section 3.3.2.

sequences length Number of queries issued by a user to achieve a certain task (Yao, et al, 2005)

sequences frequency distribution A user’s daily sequences distribution. It represents how frequently a user performs certain task on a database. In our benchmark, uniform distribution has been selected as the default distribution method.

sequence type Management: Managers check the DB periodically to collect data for reports; HR management: HR manager adjusts employee data; Financial planning: Finance Analysts send queries to DB, in order to get customers information that qualify to certain investment programs; Customer request: Transactions requested by customer (by phone, in person, by email); Product marketing: Get general info from DB to target potential market; External service: Financial Analyst gets access to DB to check certain customer’s information like credit history for third party.

number of anomalies Total number of anomaly queries within the workload

anomaly injection methods As stated above, the uniform distribution has been chosen as the default distribution method. Technically a benchmark user can implement any distribution from what to draw anomalies. However, this thesis implements a choice of Uniform or Gaussian.

anomaly types We target three types of anomalies in our benchmark. They are *data harvesting, masquerading,* and pure *sabotage attacks.* More details on anomaly design will be explained in Section 3.3.4.

3.3 Workload Design

In benchmarking parlance, the training data (normal background) and testing data (back-ground plus anomalous events) constitute the anomaly-detector workload (Maxion & Tan, 2000). The benchmark simulates a financial banking system, in which internal employees performs various transactions against a relational database. A workload driver is used to generate transaction flows on the database. It simulates *n* concurrent database users each of which connects to the database and submits a mix of transactions. In our benchmark, we consider the bank's internal employees as the only database users. Customer's interaction with the database is irrelevant to the purpose of our benchmark and its impact on IDS's performance is negligible. The transactions are generated in the similar way as proposed in Transaction Processing over XML (TPoX) benchmark. In TPoX, all transactions are picked randomly from a set of predefined transaction templates. But in our benchmark, each role has its unique set of transaction templates to be chosen from. At run time, parameter markers in the templates are replaced by actual values drawn from configurable random value distributions. The simulated banking transactions should reflect a real-world scenario. We use a daily transactions scheduler to populate and simulate each employee's everyday transaction flow. During the process of workload population, internal employees perform various transactions against the database which create the training data for the benchmark, while the normal background of the testing data are generated the same way. The workload’s testing data is the result of injected anomaly events into normal transactions. The following sections provide details of the workload design by describing transaction templates and daily transactions scheduler design, followed by the anomaly population.

3.3.1 Financial Bank Scenario

The financial banking scenario has been numerously used in OLTP benchmark applications, such as *DebitCredit, TPC-A and TPC-B*. In our benchmark for insider threat IDS, we model a financial banking system that focus on internal employees’ behavior towards the database. We consider an event is an anomaly when an employee conducts an unwanted or unexpected manipulation to the database, which may eventually result in the breach of customer's privacy. The simulated financial bank operates out of a number of *Branches*. The benchmark is designed in such a way that the size of the bank is scalable. In other words, the number of the branches and its total employees can be configured according to testing needs. Each branch has several *Department*s and every department has more than one employee. Bank *Employees* are identified by ID number. Each employee is only allowed to work under a department. Employees perform various transactions on the database corresponding to its role (which department, he or she is in). The Bank provides three types of accounts to its customers - *SavingsAccounts*, *CheckingAccounts* and *LoanAccounts*. The bank's *Customer* is identified by its ID number. Each customer can have more than one *Account*. Each account is assigned a unique account number, account balances, and account branch. In addition, each saving and checking account interest rates, and each loan account installment amount. Every updates on the customer’s account will be recorded in a *Journal* table. The database stores information of bank transactions in the Trans log file.

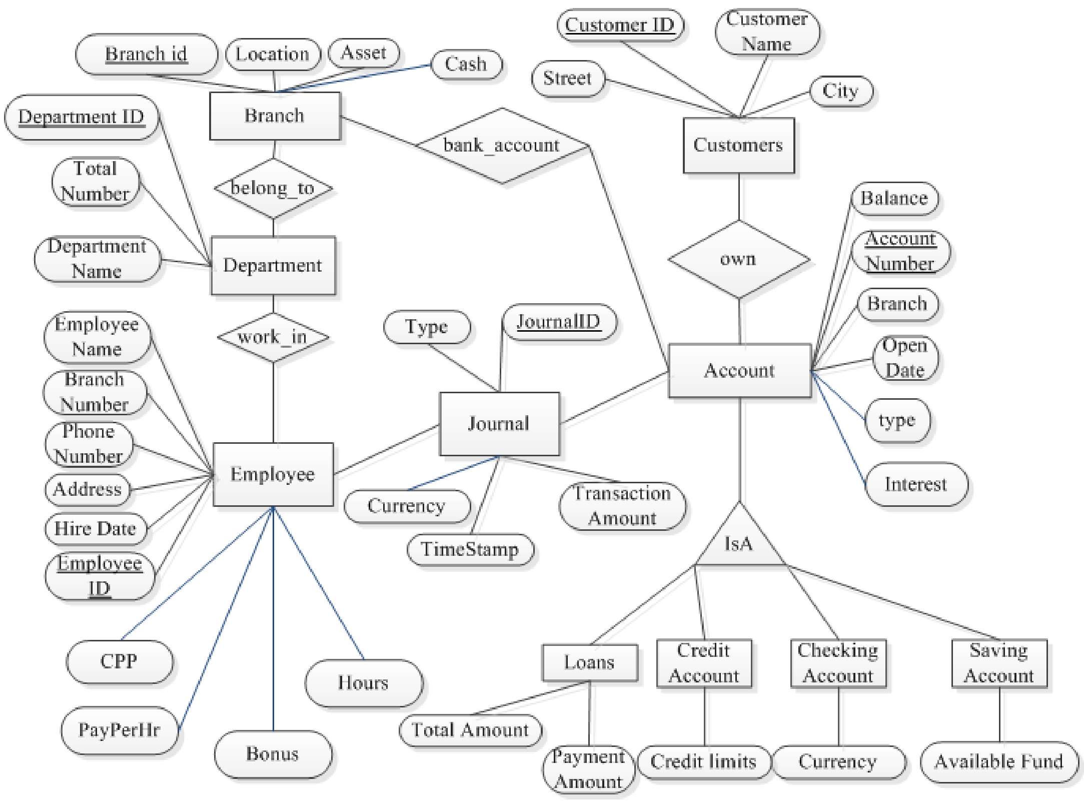


Figure 3.2 Financial Bank Entity-Relationship Diagram

The banking database maintains information on *Branch, Department, Customer, Employee, Accounts,* and *Journal*. Above in Figure 3.2 shows the Entity-Relationship (ER) diagram for a banking system workload. An entity is an object or concept about what data is stored.

In the diagram, entities, represented as rectangles, are *Branch, Department, Customer, Employee, Accounts* and *Journal.* The relationship, represented as diamonds, provides information about association among entities. The relationships *belong_to* and *bank_account* indicate the *Branch* associated with the entities *Department* and *Account*, respectively. The table *Journal* indicates the association between *Employee* and *Account* . The *IsA* relationship, depicted by a triangle, represent the "is a" relationship between low-level entities and high-level entities, which means a low-level entity is a special case of high-level entity. Low-level entities inherit properties from high-level entities.

In our case, *Loan, Credit Account, Checking Account* and *Saving Account* are low-level entities; they inherit properties from high-level *Account*. The attribute, denoted by ovals, indicate properties of the entities or relationships to which they are connected. The underlined attributes are key property, which are uniquely identifying attributes for the entities (Dietrich et al, 1992). According to the proposed ER diagram, we can map it to a database scheme as listed below.

Table 2

Bank Database Schema

Branch(branch_id,location,asset)

Department(department_id,name,emp_totoal,branch_id)

Employee(employee_id,name,street,city,province,title,department_id,hire_date,salary,CPP, PayPerHr,Bonus,Hours)

Customer(customer_id,name,phone,street,profession,income,city,province,DateOfBirth)

Account(id,customer_id,open_date,branch_id,type,interest,balance)

CheckingAccount(id, currency)

CreditAccount(id, available_fund,last_payment)

LoanAccount(id, InstallmentAmount,penalty)

SavingAccount(id,available_fund,currency)

Journal(journal_id,employee_id,account_id,transamount,timestamp,type)

The schema for the relational data model representation of the bank is show in Table 2. The relations for *Branch, Department, Customer, Employee, Accounts, Journal,* and *Trans* include the attributes associated with the entities from the ER diagram, in which the underlined attribute are the primary key for its relation. In addition, the *Branch* and *Department* relations include the attribute branch_id, which indicates the associated branch as given by the relationship *belong_to*. Likewise, the attribute employee_id included in relation *Employee* and *Transaction* indicates the relationship *update* from the ER diagram. The primary keys of the relations are given by the corresponding key attributes from the ER diagram. The *Journal* relation contains the attributes corresponding to the key attributes of the *Employee* and

Account entities and the relationship's descriptive attributes TimeStamp and TransactionAmount. The records for all the relations are populated according to the ***database size*** parameters, which can be achieved through the use of a data filler program. The parameters can be configured in a way that the total number of each table's records may vary according to testing needs.

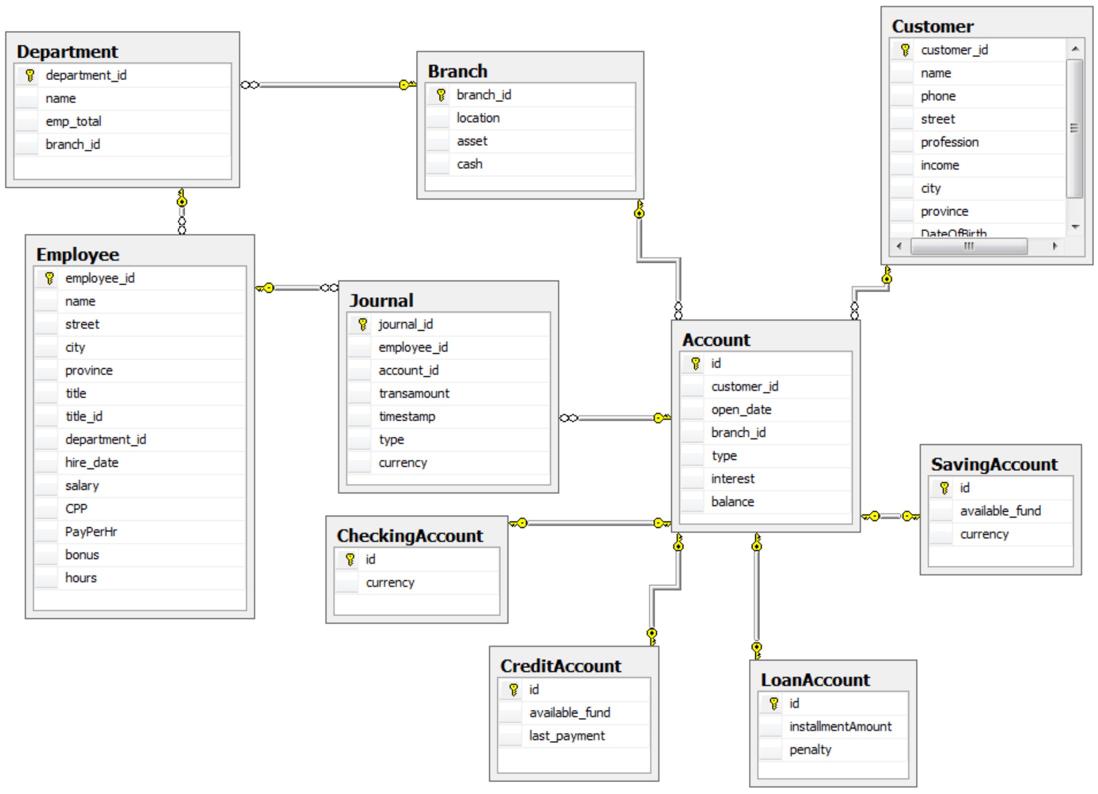


Figure 3.3 Tables' Relationships in a Bank Database Implemented in MS SQL Server

The database will be generated on Microsoft SQL Server 2008. Figure 3.3 shows the database diagram that depicts relationship between tables. Tables represent the entities in Database schema.

3.3.2 Transaction Template

In the realm of database systems, a transaction is a sequence of operations performed as a single logical unit of work. Usually, a transaction consists of several SQL commands, which are the combination of queries, inserts, updates and deletes. In our benchmark, we consider each SQL command as a single transaction. Users perform those transactions within the database to achieve a certain task. Database users execute various manipulations in a database according to his/her role. A role defines what a user can and cannot do within a database, and multiple users may share the same role. In our benchmark, we define that each role represents a unique department; employees who work in the same department share the same role, which means several employees would be entitled the same security clearance on the database if they are in the same department. In the real world, for the efficiency of managing its employees, companies create broad security clearance for each department (role) instead of each employee. That creates opportunities for employees to conduct unwanted manipulations to the database that may result in the breach of its customers' privacy.

As showed in table 3, we provide twenty roles in our benchmark. Each of these roles falls under one of four business functions – Human Resource, Marketing, Accounting, and Customer Service. In other words, up to twenty departments could exist in the banking scenario.

Table 3

Business Functions and Their Roles

Business Function	*Roles*
Human Resource Function	Payroll Clerk, Recruiter, Compensation Analyst, Benefit Clerk, Employment and organizational development director,
Marketing Function	Product development specialist, Telemarketing Representative,
Accounting Function	Cost Accountant, General Accountant, Account Clerk, Foreign Exchange Dealer, Auditor, Fraud detection specialist,
Customer Service Function	Vault Teller, Teller, Sales representative, Credit Clerk, Mortgage Loan Officer, Loan Reviewer, Mortgage Closer,

We assume employees that belong to the same department perform the same type of transactions in the database. In our benchmark, we predefine a set of transaction templates for each role. In each template, we use wildcard character '%' to represent a data value in the SQL command, which will be replaced by actual values drawn from configurable random value distributions. We will discuss each role's duty and its corresponding transaction templates in the following section.

1) Human Resource Function (HR)

Human Resource employees need to keep track on employment history in order to keep reference for future decision making. That means they will access specific attributes in the *Employee* table, which store employees' detailed information. Presumably, they log onto the database a few times and stay for short periods of time per day whenever a task requires the employee to gather information from the database. Under the human resource function, there are five roles – payroll clerk, recruiter, compensation analyst, benefit clerk, and employment and organization development director.

a) Templates for the Payroll Clerk Role

The payroll clerk is accountable for collecting timekeeping information, incorporating a variety of deductions into a periodic payroll, and issuing pay and pay-related information to employees.

i. Checking employee's working hour

69. Select hours from Employee where employee_id=’%’

ii. Making monthly payment

70. Select employee_id, PayPerHr*hours from Employee

iii. Issuing tax form

71. Select employee_id, salary from Employee

iv. Calculating over time

72. Select Bonus from Employee

b) Templates for the Recruiter Role

The recruiter is in charge of recruiting new employees for the bank. That means he/she will access the *Employee* table to create new records and update the corresponding attribute in *Department* table.

i. Recruiting a New Employee

01. SELECT department_id FROM Department WHERE name='%' AND branch_id='%'
02. SELECT max(employee_id) FROM Employee
03. INSERT INTO Employee (employee_id, name, street, city, province, title, department_id, hire_date) VALUES('employee_id','name','street','city','province','title','department_id','hire_date')
04. UPDATE Department SET emp_total=emp_total+1

c) Templates for the Compensation Analyst Role

Compensation analysts monitor salaries spend in order to optimize the investment in human resources. They also participate in the implementation of various components of the organization's pay and rewards program, which include base pay and bonuses.

i. Model Salary Administration System

11. SELECT TOP 10 * FROM Employee ORDER BY salary
12. SELECT TOP 10 * FROM Employee WHERE department_id='%' ORDER BY salary
13. SELECT SUM(salary) AS TotalSalary FROM Employee WHERE department_id='%'
14. SELECT Employee_id, name, salary, title FROM Employee WHERE salary>%

ii. Retrieving Employee Information.

05. SELECT * FROM Employee WHERE employee_id = ‘%’

15. SELECT * FROM Employee WHERE province=’%’

iii. Organizing , coordinating the accurate payment of sales bonus according to their signed compensation plan

73. Update Employee Set bonus =’%’

iv. Maintain and correct sales performance to ensure accurate bonus payment

72. Select bonus from Employee

d) Templates for the Benefit Clerk Role

A benefit clerk should help employees understand the benefits package offered by the employer. When employees have questions regarding their benefits, the benefit clerk is usually the first resource within the organization to deal with this issue.

i. Answers employee’s questions

05. Select * From Employee Where Employee_id=’%’

ii. Records employee enrollment in benefit programs: retirement, pension and health.

75. Update Employee Set CPP=’%’

e) Templates for the Employment and Organizational Development Director Role

The employment and organizational development director facilitate Human Resources functions including promoting and transferring employees within the bank.

i. Promoting an Existing Employee within a Department

05. SELECT * FROM Employee WHERE employee_id = '%'

06. UPDATE Employee SET salary= '%' WHERE employee_id='%'

ii. Transferring Employee Between Different Departments

06. SELECT * FROM Employee WHERE employee_id = '%'

07. UPDATE Employee SET salary= '%' WHERE employee_id='%'

08. UPDATE Department SET emp_total=emp_total-1 FROM Department, Employee WHERE Department.department_id= Employee.department_id AND Employee.employee_id='%'

09. UPDATE Employee SET department_id='%' WHERE employee_id='%'

10. UPDATE Department SET emp_total= emp_total+1 FROM Department, Employee WHERE Department.department_id= Employee.department_id AND Employee.employee_id='%'

iii. Layoff Management Procedures

07. UPDATE Department SET emp_total=emp_total-1 FROM Department, Employee WHERE Department.department_id= Employee.department_id AND Employee.employee_id='%'

11. DELETE * FROM Employee WHERE employee_id='%'

2) *Marketing Function*

The marketing employees are responsible for the coordination of all marketing, social media and networking activities of the Bank. Under the marketing function, there are two roles – product development specialist, and telemarketing representative.

a) Templates for the Product Development Specialist Role

In order to set up an effective marketing strategy, the product development specialists need to analyze their target market in general, as well as keeping track of customer's contact information to collect feedback. They analyze the demography of its customers in order to develop various marketing concepts, objectives, materials and advertising programs.

i. Review the bank's existing products or services and research ways to enhance them

76. Select * From Account Where type ='%'

ii. Design and develop product, promotional programs and product brochures

16. SELECT COUNT(*) FROM Customer WHERE profession ='%'
17. SELECT COUNT(*) FROM Customer WHERE province='%'
18. SELECT COUNT(*) FROM Customer WHERE DateOfBirth >'%'
19. SELECT COUNT(*) FROM Customer WHERE income>'%'
20. SELECT TOP 100 * FROM Customer AS T1, Account AS T2 WHERE T1.customer_id=T2.customer_id ORDER BY T2.Balance
21. SELECT * FROM Account Where open_date>x AND open_date<y

iii. Conduct surveys, market studies and research upon customer satisfaction in order to develop the project concept

24. SELECT * FROM Customer WHERE DateOfBirth >'%'
25. SELECT * FROM Customer WHERE income>'%'

b) Templates for the Telemarketing Representative Role

A telemarketing representative contacts customers on behalf of the bank to promote the bank's products. He/she interacts with the customers on possible solutions to their different financial problems.

i. Look up the contact information from the Customer table in order to get in touch with an individual customer; to provide possible solutions to the customers' different financial problems.

22. SELECT phoneNo FROM Customer as T1, Loan Account asT2 WHERE T1.customer_id=T2.customer_id AND T2.penalty>'%'

23. SELECT phoneNo FROM Customer as T1, Credit Account asT2 WHERE T1.customer_id=T2.customer_id AND T2.penalty>'%'

3) Accounting Function

Accountants need access to every detailed transaction (every table) to make different accounting statements. The bank accountant is responsible for preparing or assisting in the preparation of financial statements at the end of each month and quarter. This allows an investor to follow the bank's lending and investing activities more easily.

An auditor monitors the operations of the bank to ensure its compliance with industry guidelines and adherence to measures that deter fraud. In an industry frequently considered highly competitive, a bank auditor reviews the general and specific aspects of daily practices to guarantee her bank remains competitive and maintains the integrity expected by its customers. They scrutinize every practice from teller transactions through the security of the bank's vaults and courier services. Another typical area of concern for bank auditors is compliance with regulations and guidelines governing lending practices and disclosure procedures. Since the industry guidelines are frequently updated and revamped, it is important for the bank to be in total compliance in their operations to avoid penalties and properly secure customer deposits.

Under the accounting function, there are six roles – cost accountant, general accountant, account clerk, foreign exchange dealer, auditor, and fraud detection specialist.

a) Templates for the Cost Accountant Role

The cost accountant is accountable for the ongoing analysis of target costing products and margin analysis.

i. Looking after product costing

78. SELECT SUM(balance*interest) AS interestIncome FROM Account

ii. Allocation of common costs or overheads with system process

46. Select SUM(salary) from Employee

b) Templates for the General Accountant Role

A general accountant's duties include preparing journal entries, maintaining balance sheet schedules and assisting with monthly closings and account analysis. She/he also in charge of assisting in the preparation of various financial statements, which include an income statement, balance sheet, cash flow statement and the statement of retained earnings.

i.Cash Position Report

41. SELECT SUM (transamount) FROM Journal WHERE timestamp=’today’ AND type=’credit’

42.SELECT SUM (transamount) FROM Journal WHERE timestamp=’today’ AND type=’transfer’

40. SELECT SUM (transamount) FROM Trans WHERE timestamp=’today’ AND type=’debit’

Income Satement

43. SELECT SUM(balance*interest) AS interestIncome FROM Account WHERE type=’loan’

44. SELECT SUM(balance*interest) AS interestExpense FROM Account WHERE type=’saving’

45. SELECT SUM(balance*interest) AS interestExpense FROM Account WHERE type=’checking’

46. SELECT SUM(salary) FROM Employee

ii. Balance Sheet

47. SELECT SUM(balance) FROM Account WHERE type =’saving’

48. SELECT SUM(balance) FROM Account WHERE type =’checking’

49. SELECT SUM(balance) FROM Account WHERE type=’credit’

50. SELECT SUM(asset) FROM Branch

iii. Cash Flow

43. SELECT SUM(balance*interest) AS interestIncome FROM Account WHERE type=’loan’

44. SELECT SUM(balance*interest) AS interestExpense FROM Account WHERE type=’saving’

45. SELECT SUM(balance*interest) AS interestExpense FROM Account WHERE type=’checking’

c) Templates for the Account Clerk Role

An account clerk is responsible for monitoring the bank's daily transaction and maintaining account records, including posting and adjusting entries in *Journal*.

i. Preparing & Posting Adjusting Entries

51. UPDATE Journal SET transamount='%' WHERE account_id='%' AND timestamp='%'
52. UPDATE Journal SET trans_balance='%' WHERE account_id='%' AND timestamp='%'

ii. Monitor Daily Transaction

53. SELECT * FROM Journal WHERE timestamp='today'
54. SELECT * FROM Journal WHERE type='%'

d) Templates for the Foreign Exchange Dealer Role

The foreign exchange dealer engages in financial market transactions and evaluates investment opportunities in currencies. A foreign exchange dealer who uses corporate funds to buy and sell currencies is referred to as a proprietary trader.

i. Accessing bank foreign exchange deposit balance

80. Select * form Journal where type ='debit' and currency='%'

ii. Monitoring foreign currencies transactions

81. Select * from Journal where type='credit' and currency ='%'
82. Select SUM(transamount) from Journal where currency='%'

e) Templates for the Auditor Role

Bank auditors need the access to the database to audit the large sums reported by accountants in their statements, and cash flow (cash inflow/outflow): employee's pension, insurance, salary, and any cash flow occurred during transactions**.** They run a number of tests, and compare statements to balance sheets to verify all documents contain correct information.

i. Auditor important accounts

55. SELECT TOP 10 * FROM Account, SavingAccount WHERE Account.id=SavingAccount.id ORDER BY balance
56. SELECT TOP 10 * FROM Account, SavingAccount WHERE Account.id=SavingAccount.id AND currency =’%’ ORDER BY balance
57. SELECT TOP 10 * FROM Account, Checking Account WHERE Account.id=SavingAccount.id ORDER BY balance

ii. Verify that processing charges are debited accordingly to borrower’s ledger

58. SELECT account_id FROM Journal WHERE transamount>’%’

59.SELECT account_id, transamount FROM Journal WHERE transamount>’%’

60. SELECT id, balance, installment FROM Account, Loan Account WHERE Account.id=LoanAccount.id AND balance>’%’

f) Templates for the Fraud Detection Specialist Role

The fraud detection specialists are responsible for analyzing and preventing dollar loss to the bank by monitoring the various reports for potential fraud activity. They are also responding to inbound calls from customers, call centers, merchants, authorization centers and police worldwide to verify questionable transactions.

i. Monitoring various transactions for potential fraud activity

47. SELECT SUM(balance) FROM Account WHERE type='saving'
48. SELECT SUM(balance) FROM Account WHERE type='checking'
49. SELECT SUM(balance) FROM Account WHERE type='credit '
50. SELECT SUM(asset) FROM Branch

ii. Contacting customers to verify transactions

83. Select phoneNo from Customer where customer_id='%'

iii. Accessing to customers' records in order to confirm questionable transactions

84. Select * from Account where customer_id = '%'

4) Customer Service Function

Customer service employees need the access to single customer record in order to provide personal service. Customer service is the main channel that customers have to communicate with the banking system. Under the customer service function, there are seven roles – vault teller, teller, sales representative, credit clerk, mortgage loan officer, loan reviewer, and mortgage closer.

a) Templates for the Vault Teller Role

A vault teller is in charge of preparing the bank for that day's transactions. She/he counts and records the night deposits, the money transactions from the day before and verifies the vault's money supply count that was made the day before. They are also responsible for ordering currency from the national bank, monitoring money supply and recording deposits.

i. Verifying cash deposit

88. Select cash from Branch where id='%'

ii. Ajusting cash balance for the branch

89. Update Branch Set cash='%'

b) Templates for the Teller Role

Tellers have a wide variety of duties such as check cashing to deposits/withdraws, making loan payments. Banking facilities often offer many services, information to clients. When tellers meet their clients, they will assist the clients with various services the bank has to offer, provide information, and redirect the customers to the appropriate department. The basic duties as a teller are: deposit, withdraw, transfer, check account balance, check cashing, bill payments, currency exchange, processing loans. Bank tellers balance their receipts and payments at the end of each day. High volume of transaction traffic is expected for tellers.

i. Deposit

26. UPDATE Account SET balance=balance+'%' WHERE account_id='%'

ii. Withdraw

28. UPDATE Account SET balance= balance-'%' WHERE account_id='%'

iii. Transfer between accounts

28. UPDATE Account SET balance= balance-'%' WHERE account_id='%'

26. UPDATE Account SET balance=balance+'%' WHERE account_id='%'

iv. Closing and Opening of Bank Accounts

30. INSERT INTO Customer (customer_id, name, street, city, province, postal_code, profession, DateOfBirth, income) VALUES ('customer_id', 'name', 'street', 'city', 'province', 'postal_code', ' profession', 'DateOfBirth', 'income')

31. INSERT INTO Account (id, customer_id, open_date, branch_id, type, interest, balance)

VALUES ('id','customer_id','open_date','branch_id','type','interest', 'balance')

32. INSERT INTO Checking Account (id, currency) VALUES ('id', 'currency')

33. UPDATE Checking Account SET balance= 0 WHERE id='%'

v. Check Account Detail

35. SELECT * FROM Account WHERE id='%'

vi. Cashing Checks

26. UPDATE Account SET balance=balance+'%' WHERE account_id='%'

vii. Currency Exchange

38. UPDATE Account SET balance = '%' WHERE currency='%' AND account_id='%

viii. Accepting Loan Payments

39. SELECT * FROM Account, Loan Account WHERE Account.id=LoanAccount.id AND id='%'

ix. Balance their receipts and payments at the end of each day

40. SELECT SUM (transamount) FROM Journal WHERE timestamp='today' AND type='debit'
41. SELECT SUM (transamount) FROM Journal WHERE timestamp='today' AND type='credit'

c) Templates for the Sales Representative Role

A sales representative works as a salesperson and customer services agent, directly with potential customers to determine their financial viability or needs and sell them a variety of financial services. The sales representative need to be familiar with the bank's offered checking or saving accounts, mortgages or securities portfolios in order to deduce which would best fit the prospective customer.

i. Reviewing customers' personal accounts to determine what banking products might be appropriate for customers

64. Select * from Customer Where customer_id='%'

ii. Answering customers' questions about products, prices, availability, product uses, and credit terms

77. Select distinct type from Account

d) Templates for the Credit Clerk Role

A credit clerk processes applications of individuals applying for credit. She/he also establishes credit limit, considering such factors as applicant’s assets, credit experience and personal references, based on predetermined standards.

i. Accessing the applicant’s records in order to determine credit limit

65. SELECT * FROM Customer WHERE Customer_id=’%’
66. SELECT * FROM Account WHERE customer_id=’%’
85. SELECT * FROM Account, Loan Account WHERE id=’%’ AND type=’credit’

ii. Creating new credit records for the accepted applicants

30. INSERT INTO Customer (customer_id, name, street, city, province, postal_code, profession, age, income) VALUES (‘customer_id’, ’name’, ’street’, ’city’, ’province’, ’postal_code’, ‘ profession’, ‘age’, ‘income’)

31. INSERT INTO Account (id, customer_id, open_date, branch_id, type,interest,balance)

VALUES (id, customer_id, open_date, branch_id, type, interest, balance)

86. INSERT INTO Credit Account (id ,available_fune, last_payment)

VALUES(‘id’, available_fund , last_payment)

iii. Adjust incorrect credit charges and grant extensions of credit on overdue accounts

87. Update   Credit Account Set   available_fund=’%’ where id=’%’

e) Templates for the Mortgage Loan Officer Role

To increase the amount of transactions through their facility, loan officers assist individuals who seek loans, and act as a liaison to creditors and borrowers. Originators aid clients when applying for loans, as well as ensure that the necessary information is provided to both the creditor and the borrower.   These professionals must be able to advise clients experiencing difficulties receiving loans, in addition to suggesting the best possible option(s) for each client. Other duties include creating loan accounts, checking loan type, tracking interest rates and updating the *LoanAccount* table.

i. Analyze applicants' financial status, credit, and property evaluations to determine feasibility of granting loans

65. SELECT * FROM Customer WHERE Customer_id=’%’

66. SELECT * FROM Account WHERE customer_id=’%’

39. SELECT * FROM Account, Loan Account WHERE id=’%’ AND type=’loan’

ii. Explain to customers the different types of loans and credit options that are available, as well as the terms of those services

77. Select interest from Account where type= ’loan’

iii. Review and update loan files

Take information from the prospective borrower and complete the loan application form

29. INSERT INTO Customer (customer_id, name, street, city, province, postal_code, profession, age, income) VALUES ('customer_id', 'name', 'street', 'city', 'province', 'postal_code', ' profession', 'age', 'income')

30. INSERT INTO Account (id, customer_id, open_date, branch_id, type,interest,balance)

VALUES (id, customer_id, open_date, branch_id, type, interest, balance)

31. INSERT INTO Loan Account (id ,installmentAmount, interest)

VALUES('id', installmentAmount , interest)

f) Templates for the Loan Reviewer Role

A loan reviewer performs loan review duties in the following areas and write reports on results of analysis performed on the various loan types, denied and withdrawn mortgage loans, and any other analysis requested.

i. Reviewing various loan accounts

90. SELECT id, interest, balance FROM Loan Account

ii. Tracking all corrections identified in risk reports and ensures corrections are reported back to the appropriate party

68. UPDATE LoanAccount SET penalty='%' WHERE id='%'

g) Templates for the Mortgage Closer Role

A mortgage closer schedules loan closing and compiles and types closing documents. She/he also in charge of reviewing approved mortgage loan to determine conditions that must be met prior to closing.

i. Preparing and verifying closing documents during the completion of real estate transactions

39. SELECT * FROM Account, Loan Account WHERE Account.id=LoanAccount.id AND id=’%’

ii. Verifying loan interest and principal payment, and closing costs

98. Select * from Journal where account_id=’%’

99. Select SUM(transamount) from Journal where account_id=’%’

100. Select penalty LoanAccount where id=’%’

iii. Updating loan information in log and on government reporting forms

94. Update LoanAccount Set penalty=0 where id=’%’

3.3.3 Daily Transactions Scheduler

In the benchmark, we use a daily transactions scheduler to populate each employee's everyday transaction flow. A scheduler dynamically generates each employee’s transaction for a day. That means each employee’s everyday transaction length is arbitrary. The scheduler stores a database access pattern according to each role’s characteristics.

In the relational database system environment, accesses to the database come from various application sources. The combination of transaction and query accesses

Table 4

Daily Transactions Scheduler Fields

Fields	**Description**
timestamp	Date and time when the transaction happens
timespan	Time spend on database per login
logins	How many times login to the database
empId	Employee identification
role	Roles that employee possessed
FirstLogin	The time an employee first login to the database a day

generates all possible access patterns (Sacco and Schkolnik, 1986; Kearns and DeFazio, 1989). In a financial bank background, bank employees retrieve private data that are stored in a database system through different banking applications. Data is being accessed according to different transactions or queries that an employee performs against a database system.

In Table 4, it shows the fields that store in a daily transactions scheduler record. According to the nature of the jobs and different transaction templates being used, each business function has its unique database access pattern. In the following we will discuss in detail the different database access patterns categorized by the business functions as we mentioned in section 3.3.2.

a) Human Resource Function

A human resource employee logins several times a day to the database, whenever there is a task involving usage of employee records. She/he stays on the database for a short period of time approximately 30 minutes to 1 hour. Therefore, we can assign ranges for the fields: time span, logins and start time in the daily transactions scheduler according to each role's characteristic. For *Human Resource* role, we assign each field's ranges as followed: timespan [30 – 60 minutes]; logins [1 to 2 times]; FirstLogin [8:00 – 16:00].

At run time, an instance of the daily transaction scheduler is generated for each employee. The fields of the scheduler instance are assigned with actual random values within each field's range based on a poisson distribution.

b) Marketing Function

A marketing employee logins frequently to the database a day. She/he stays for a longer period of time. Therefore we can assign corresponding fields of the daily scheduler with following ranges: timespan [60-120 minutes]; logins [1 to 3 times]; FirstLogin [8:00 – 16:00].

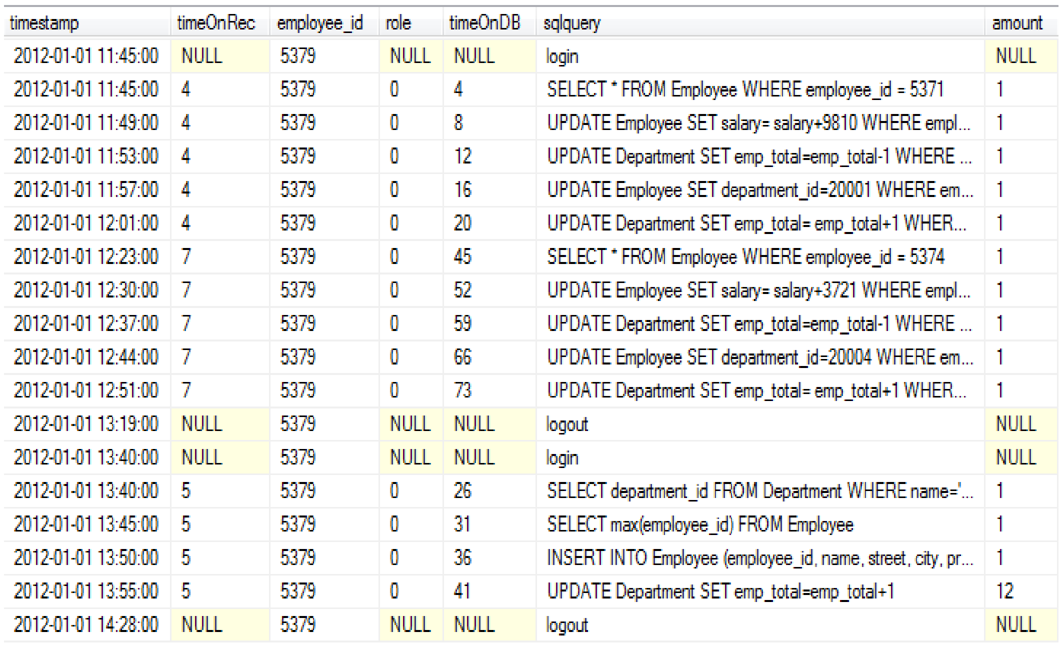
c) Accounting Function

An accounting employee is a database heavy user similar to the teller. She/he normally logins to the database when work starts and stays logged in for a long period of time. Heavy database usage can be assumed at the end of the month or quarter. Therefore we can assign corresponding fields of the daily scheduler with the following ranges: time span [180-240 minutes]; logins [2]; FirstLogin [8:00].

d) Customer Service Function

A customer service employee logins twice to the database a day, when each shifts starts (morning and afternoon). She/he stays logged in during her/his shift. Therefore we can assign corresponding fields of the daily scheduler with following ranges: timespan [180-240 minutes]; logins [2]; FirstLogin [8:00].

For further explaining the purpose of the daily transaction scheduler, an example is given by using the scheduler to populate a transaction flow for a human resource employee. Given the random value drawn from the field ranges are timespan [30-60], logins [2] and FirstLogin [11:45], the example is listed in Figure 3.4.



timestamp	timeOnRec	employee_id	role	timeOnDB	sqlquery	amount
2012-01-01 11:45:00	NULL	5379	NULL	NULL	login	NULL
2012-01-01 11:45:00	4	5379	0	4	SELECT * FROM Employee WHERE employee_id = 5371	1
2012-01-01 11:49:00	4	5379	0	8	UPDATE Employee SET salary= salary+9810 WHERE empl...	1
2012-01-01 11:53:00	4	5379	0	12	UPDATE Department SET emp_total=emp_total-1 WHERE ...	1
2012-01-01 11:57:00	4	5379	0	16	UPDATE Employee SET department_id=20001 WHERE em...	1
2012-01-01 12:01:00	4	5379	0	20	UPDATE Department SET emp_total= emp_total+1 WHER...	1
2012-01-01 12:23:00	7	5379	0	45	SELECT * FROM Employee WHERE employee_id = 5374	1
2012-01-01 12:30:00	7	5379	0	52	UPDATE Employee SET salary= salary+3721 WHERE empl...	1
2012-01-01 12:37:00	7	5379	0	59	UPDATE Department SET emp_total=emp_total-1 WHERE ...	1
2012-01-01 12:44:00	7	5379	0	66	UPDATE Employee SET department_id=20004 WHERE em...	1
2012-01-01 12:51:00	7	5379	0	73	UPDATE Department SET emp_total= emp_total+1 WHER...	1
2012-01-01 13:19:00	NULL	5379	NULL	NULL	logout	NULL
2012-01-01 13:40:00	NULL	5379	NULL	NULL	login	NULL
2012-01-01 13:40:00	5	5379	0	26	SELECT department_id FROM Department WHERE name='...	1
2012-01-01 13:45:00	5	5379	0	31	SELECT max(employee_id) FROM Employee	1
2012-01-01 13:50:00	5	5379	0	36	INSERT INTO Employee (employee_id, name, street, city, pr...	1
2012-01-01 13:55:00	5	5379	0	41	UPDATE Department SET emp_total=emp_total+1	12
2012-01-01 14:28:00	NULL	5379	NULL	NULL	logout	NULL

Figure 3.4 A HR Employee Transaction Log For a Day Example

3.3.4 Types of Insider Attacks

Numerous definitions have been proposed for the term insider attack. According to Maybury et al. (2005), malicious insider is one motivated to adversely impact an organization's confidentiality, integrity, and or availability. Schultz (2002) defines an insider attack as "the intentional misuse of computer systems by users who are authorized to access those systems and networks". Aleman-Mezal, et al. (2005) asserts that insider threat refers to the "potential malevolent actions by employees within an organization, a special type of which relates to legitimate access of document". The above definition involves the notion of assigned privileges to an insider. So general definition of insider has privileges through which he/she can access different information in his/her organization (Mun, H., Han, K., Yeun, C., & Kim, K., 2008).

Anderson (1980) first proposed the classification for insiders who may misuse the IT systems into *masqueraders, clandestine users,* and *misfeasors.* Salem, et al. (2008) defines a malicious insider to be two classes of malfeasant users; *traitors* and *masqueraders*. However, both classifications only characterized the type of users and not the actual attacks or the purpose of the attacks. For the purposes of this benchmark, we categorized anomalies according to the types of the insider attacks instead of actual attackers; we define anomaly to be three classes of attacks: *Data-harvest, Masquerade,* and *Sabotage.*

a) Data-harvest

Data-harvest is probably one of the most common insider attacks. The attacker could be a legitimate database user within an organization who abuses his/her privilege to gather valuable information of its customer and perhaps sell it to a third party. An example of the attack would be a bank employee who exports a list of its customers' demographic information and sells it to an insurance company. This type of attack is probably the hardest to detect. During the attack, we can assume massive data would being accessed; the attacker probably need to frequently access sensitive data and logins to the database outside of his/her working hours. In our benchmark, we define sensitive data that are stored in *Loan Account Table*, *Credit Account Table*. The transaction templates of the attacks are listed as followed.

i. Access massive Loan Account Information

SELECT * FROM LoanAccount

SELECT * FROM Account, CustTable WHERE   Account.type='loan'

SELECT * FROM Loan Account WHERE penalty > 100

ii. Access massive Credit Account Information

SELECT * FROM CreditAccount

SELECT * FROM Account, CustTable WHERE Account.customer_id = Customer.customer_id AND Account.type='credit'

SELECT * FROM Account WHERE balance>100 AND type='credit'

b) Masquerade

According to Salem, et al. (2008), the most familiar example of an insider is a masquerader; an attacker who succeeds in stealing in legitimate user's identity and impersonates another user for malicious purpose. Credit card fraudsters are perhaps the best example of masqueraders. In our benchmark, we define a masquerading attacker is the one who illegally use other database user’s identity to perpetrate malicious actions. The attackers are drawn from the same probability distribution as of normal workload background, but with role information negated.

3) Sabotage

Sabotage is probably one of the severest insider attacks that an organization can encounter. A sabotage attack could be launched by a disgruntled former employee or an unsatisfied current employee who seek revenge to the organization. The attacker uses his/her knowledge of the company and the infrastructure of the database system to perpetrate their malicious actions, causing negatively effects on confidentially, integrity or availability of some information asset (Maybury,2005). One of the typical examples of this attack in our benchmark as listed bellowed, is deleting a journal log and making the record files untraceable.

```
DELETE FROM Journal WHERE transamount >1000
```

Chapter 4

Implementation

In this chapter, the procedure of implementing the benchmark is explained in section 4.1. In the second part of the chapter, we discuss the techniques that have been used to implement two testing intrusion detection systems – Bayesian Network IDS (An&Jutla, 2006) and RBAC Classifier (Bertino, et.al, 2005).

4.1 Benchmark Implementation

As showed in Figure 4.1, in order to create the workload, the benchmark includes *Employee Profile Generator*, *Workload Driver, Anomaly Generator, Anomaly Injector,* and *Database Populator*. We chose Microsoft SQL Server as our Database

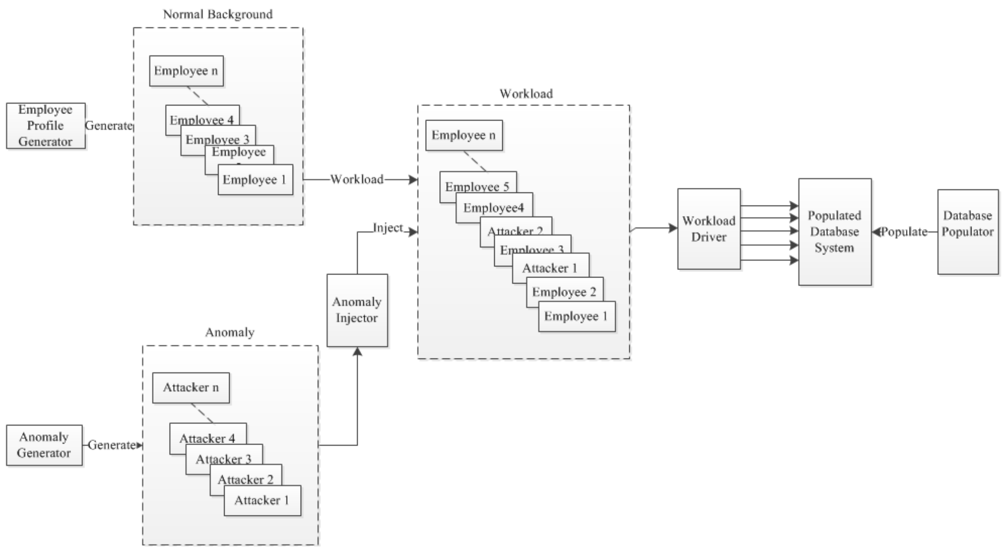


Figure 4.1 Benchmark Workload Flow

Management System (DBMS) to build our financial bank infrastructure, since MS

SQL Server is one of the popular DBMS used by organizations. We used Java for the implementation of *Employee Profile Generator, Anomaly Generator, Anomaly Injector* and *Workload Driver* for its availability and platform independence. All our experiments were conducted with Java 6.5 and MS SQL Server 2008 R2, but earlier versions of both software should work as well.

4.1.1 Database Populator Implementation

Database populator is a benchmark component to populate records in the database; it has been implemented by using Java 6.5. We used the Java Database Connectivity (JDBC) driver provided by Microsoft to make connections to MS SQL Server. JDBC driver is a software component enabling a Java application to interact with a database, such as Java SE Technologies (2012). To make connection with an individual database, JDBC requires drivers for each database. Type 4 JDBC driver for MS SQL Server was used; it provides database connectivity through the standard JDBC application program interfaces (APIs). The JDBC driver gives out a connection to our benchmark banking database and implements the protocol for transferring the query and result between the database populator and the database.

In a Java application, the JDBC driver requires to specify *driver class, driver location,* and *JDBC URL format* in order to make a connection to the database. An example of JDBC driver being used in our benchmark is listed in Figure 4.2. As shown in the below, we log into the database, *Financial Bank*, as system administrator (represented as’ sa’ in Figure 6). The system administrator’s credentials are predefined.

```
//Load and register SQL Server driver
Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
//Establish a connection
  String connectionUrl="jdbc:sqlserver://localhost:1433; "+"databaseName=Financial
Bank; user=sa; password=Bm123456";
  Connection conn = DriverManager.getConnection(connectionUrl);
//Create a Statement object
Statement sql_stmt = conn.createStatement();
```

Figure 4.2 A JDBC Driver Example Used in Database Populator

The size of the database is define by four parameters – total number of roles, total number of branches, maximum employee per departments, and total number of customers. By specifying these four parameters, we can use the population component to create a desired size of database. After defining the parameters, the method *initialize (*int *NumOfDepartment,* int *BranchTotal,* int *MaxEmpPerDep,* int *CustomerTotal)* is called, which creates four instances of the classes – *Branch, Department, CustTable,* and *EmpTable*. Each of these classes are in charge of populating one or a group of tables in the database.

As mentioned in Section 3.3.1, there are several tables in the Financial Bank database – *Department table, Employee table, Journal table, Trans table, Branch table, Customer table, Account table, Checking Account table, Credit Account table, Loan Account table,* and *Saving Account table.*

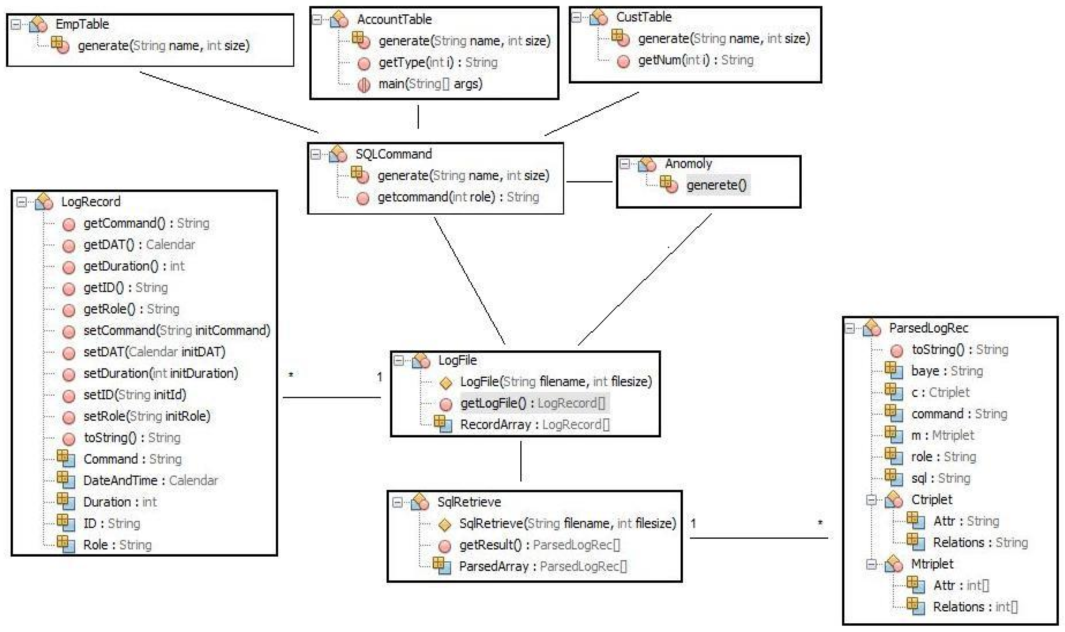


Figure 4.3 Class Diagram of Database Populator

a) Branch Class

This class has one public method: *generate (*Statement *sql_stmt*, int *BranchTotal)*. It is in charge of populating branch records into the *Branch* table according to the parameter: total number of branches.

b) Department Class

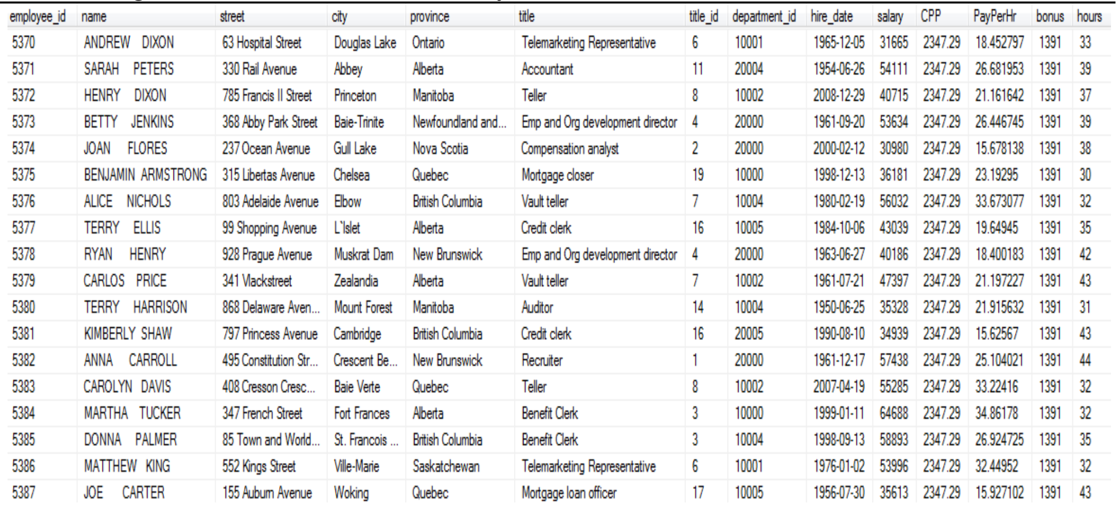
This class is in charge of generating the *Department* table, which has two method: *generate (*Statement *sql_stmt)* and *getName (*int *i)*. This class describes how many departments exist in the database, which is controlled by the total number of branches and the total number of roles. Given there are N branches and M roles, the total number of departments that exist in the database is N×M.

c) EmpTable Class

This class is used to generate records into the *Employee* Table, which stores employees' demographic information. It includes two methods: *getTitle (*int *role)* and *generate (*Statement *sql_stmt,* int *MaxEmpPerDep)*. Method *getTitle (*int *role)* returns the job title of the position an employee holds according to his/her role.

In our benchmark, we assume that one person holds only a given title in an organization. Public method *generate (*Statement *sql_stmt,* int *MaxEmpPerDep)* creates records into *Employee* table. The total number of records in the *Employee* table is determined by the number of departments that exist in the database and the parameter maximum employee per department.

Given there is at least one employee in a department and there are N departments in the database, the total number of employees lays between N and N × Maximum Employee per Department. Figure 4.4 is an example of a generated *Employee* table.



employee_id	name	street	city	province	title	title_id	department_id	hire_date	salary	CPP	PayPerHr	bonus	hours
5370	ANDREW DIXON	63 Hospital Street	Douglas Lake	Ontario	Telemarketing Representative	6	10001	1965-12-05	31665	2347.29	18.452797	1391	33
5371	SARAH PETERS	330 Rail Avenue	Abbey	Alberta	Accountant	11	20004	1954-06-26	54111	2347.29	26.681953	1391	39
5372	HENRY DIXON	785 Francis II Street	Princeton	Manitoba	Teller	8	10002	2008-12-29	40715	2347.29	21.161642	1391	37
5373	BETTY JENKINS	368 Abby Park Street	Baie-Trinite	Newfoundland and...	Emp and Org development director	4	20000	1961-09-20	53634	2347.29	26.446745	1391	39
5374	JOAN FLORES	237 Ocean Avenue	Gull Lake	Nova Scotia	Compensation analyst	2	20000	2000-02-12	30980	2347.29	15.678138	1391	38
5375	BENJAMIN ARMSTRONG	315 Libertas Avenue	Chelsea	Quebec	Mortgage closer	19	10000	1998-12-13	36181	2347.29	23.19295	1391	30
5376	ALICE NICHOLS	803 Adelaide Avenue	Elbow	British Columbia	Vault teller	7	10004	1980-02-19	56032	2347.29	33.673077	1391	32
5377	TERRY ELLIS	99 Shopping Avenue	L`Islet	Alberta	Credit clerk	16	10005	1984-10-06	43039	2347.29	19.64945	1391	35
5378	RYAN HENRY	928 Prague Avenue	Muskrat Dam	New Brunswick	Emp and Org development director	4	20000	1963-06-27	40186	2347.29	18.400183	1391	42
5379	CARLOS PRICE	341 Vlackstreet	Zealandia	Alberta	Vault teller	7	10002	1961-07-21	47397	2347.29	21.197227	1391	43
5380	TERRY HARRISON	868 Delaware Aven...	Mount Forest	Manitoba	Auditor	14	10004	1950-06-25	35328	2347.29	21.915632	1391	31
5381	KIMBERLY SHAW	797 Princess Avenue	Cambridge	British Columbia	Credit clerk	16	20005	1990-08-10	34939	2347.29	15.62567	1391	43
5382	ANNA CARROLL	495 Constitution Str...	Crescent Be...	New Brunswick	Recruiter	1	20000	1961-12-17	57438	2347.29	25.104021	1391	44
5383	CAROLYN DAVIS	408 Cresson Cresc...	Baie Verte	Quebec	Teller	8	10002	2007-04-19	55285	2347.29	33.22416	1391	32
5384	MARTHA TUCKER	347 French Street	Fort Frances	Alberta	Benefit Clerk	3	10000	1999-01-11	64688	2347.29	34.86178	1391	32
5385	DONNA PALMER	85 Town and World...	St. Francois ...	British Columbia	Benefit Clerk	3	10004	1998-09-13	58893	2347.29	26.924725	1391	35
5386	MATTHEW KING	552 Kings Street	Ville-Marie	Saskatchewan	Telemarketing Representative	6	10001	1976-01-02	53996	2347.29	32.44952	1391	32
5387	JOE CARTER	155 Auburn Avenue	Woking	Quebec	Mortgage loan officer	17	10005	1956-07-30	35613	2347.29	15.927102	1391	43

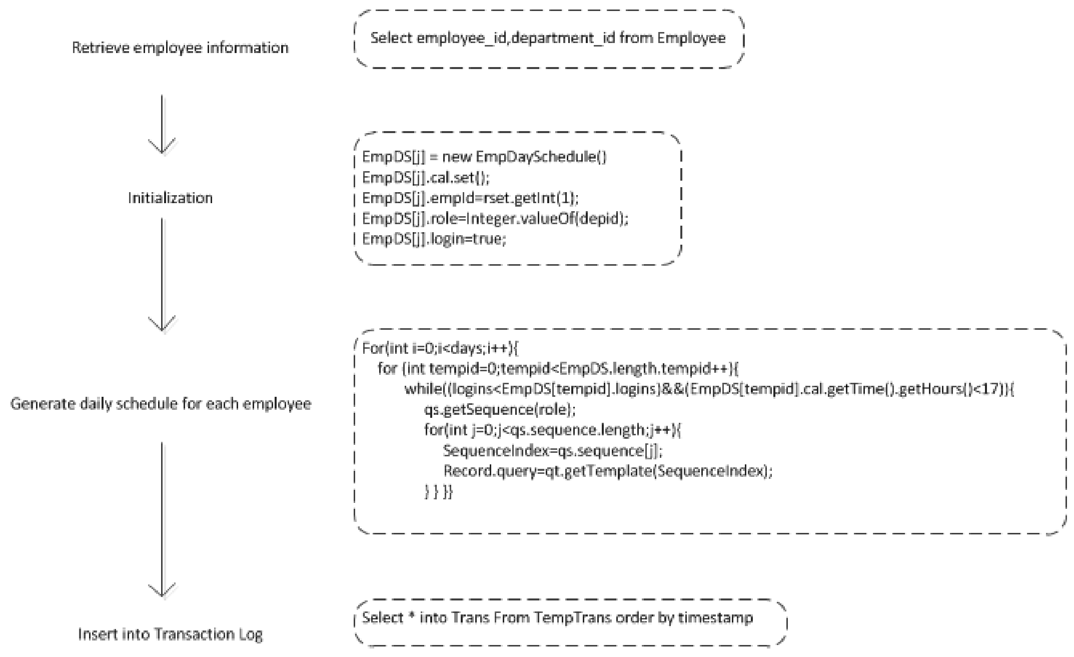
Figure 4.4 An Example of Generated *Employee* Table

d) CustTable Class

This class is used to generate the *Customer* table, which stores customers' information. The method we used to populate records in the *Customer* table is similar to the one for the *Employee* table. According to the parameter, total number of customers, we can create the desired size of the table. After the *Customer* table has been populated, a method of the AccountTable class is being called to create a group of account records for each customer. According to the type of the account record, it further distributes that record to corresponding sub-account tables.

4.1.2 Employee Profile Generator Implementation

Employee Profile Generator uses the Daily Transaction Scheduler component as we mentioned in section 3.3.3 to generate transaction traffic for all employees. In order to achieve this, the generator first retrieves employee_id and department_id from the *Employee* table, then creates an instance of Daily Transaction Scheduler class for each employee records. By creating daily transaction traffic for each employee, we can generate a daily transaction log for the financial bank database. As shown in Figure 4.4, the generator first creates the transaction flow into a temporary table called *TempTrans*. Its records are ordered by employee_id; then it inserts all the records into the *Trans* table ordered by timestamp.


Retrieve employee information
Initialization
Generate daily schedule for each employee
Insert into Transaction Log

```
Select employee_id,department_id from Employee
```

```
EmpDS[j] = new EmpDaySchedule()
EmpDS[j].cal.set();
EmpDS[j].empId=rset.getInt(1);
EmpDS[j].role=Integer.valueOf(depid);
EmpDS[j].login=true;
```

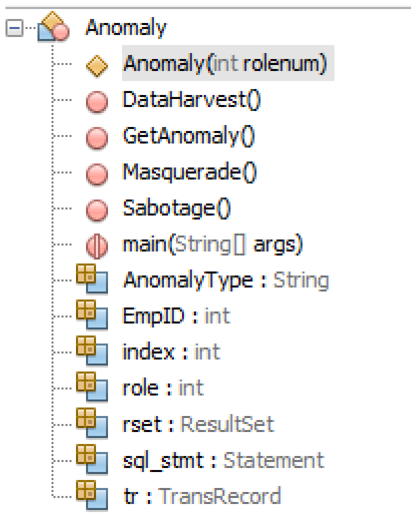
```
For(int i=0;i<days;i++){
  for (int tempid=0;tempid<EmpDS.length.tempid++){
      while((logins<EmpDS[tempid].logins)&&(EmpDS[tempid].cal.getTime().getHours()<17)){
         qs.getSequence(role);
         for(int j=0;j<qs.sequence.length;j++){
            SequenceIndex=qs.sequence[j];
            Record.query=qt.getTemplate(SequenceIndex);
         } } }}
```

```
Select * into Trans From TempTrans order by timestamp
```

Figure 4.5 Employee Profile Generator Workflow

4.1.3Anomaly Generator Implementation

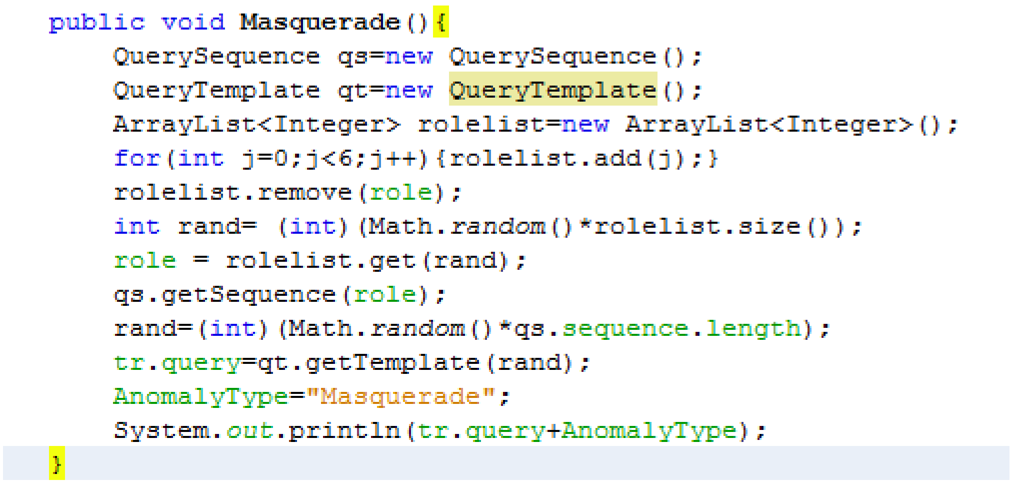
As mention in section 3.3.4, there are three types of insider attacks – data harvest, masquerade and sabotage. According to the types of the attack, we can create different anomaly records by using the generator. Figure 4.6 shows the members' view of Anomaly Generator class. In the class, we have three functions to generate anomaly transactions; each of the functions corresponds to one of the three insider attacks.



```
Anomaly
    Anomaly(int rolenum)
    DataHarvest()
    GetAnomaly()
    Masquerade()
    Sabotage()
    main(String[] args)
    AnomalyType : String
    EmpID : int
    index : int
    role : int
    rset : ResultSet
    sql_stmt : Statement
    tr : TransRecord
```

Figure 4.6 Anomaly Generator Members View

We give details of the Java code for function *Masquerade ( )* in Figure 4.7 below.



```java
public void Masquerade(){
    QuerySequence qs=new QuerySequence();
    QueryTemplate qt=new QueryTemplate();
    ArrayList<Integer> rolelist=new ArrayList<Integer>();
    for(int j=0;j<6;j++){rolelist.add(j);}
    rolelist.remove(role);
    int rand= (int)(Math.random()*rolelist.size());
    role = rolelist.get(rand);
    qs.getSequence(role);
    rand=(int)(Math.random()*qs.sequence.length);
    tr.query=qt.getTemplate(rand);
    AnomalyType="Masquerade";
    System.out.println(tr.query+AnomalyType);
}
```

Figure 4.7 Java Codes for Masquerade ( ) Function

4.1.4 Anomaly Injector Implementation

In order to create workloads for Intrusion Detection to run against, we need to inject anomaly transactions into a normal background; in our benchmark, a normal background is the employee transaction traffic we created by using the employee profile generator and the anomaly is the insider attack behaviors. In our benchmark, we provide two anomaly injection methods: uniform distribution and Gaussian distribution. We used two methods provided by Java to realize the injection. One is Math.random ( ) for uniform distribution; the other is rand.nextGaussian ( ) for Gaussian distribution.

4.1.5 Workload Driver Implementation

In the previous section, we described how to use the employee profile generator to create the normal background and the anomaly generator to create insider attacks. By injecting anomalies into normal background, we enriched the workload for intrusion detection system to test on.

The tests are conducted on the transaction log records, which stored in *Trans* table as listed in Table 5, and injected anomalies. The workload driver uses *executeQuery* ( ) from the class Statement, provided by Java, to execute SQL queries and stores the execution log in the *Trans* table. While running IDSs tests, an instance of *SQLParser* class is created to parse the SQL queries and stored in a *ParsedLogRecords* Array.

Table 5

Trans Table's Attributes Description

Attribute	*Description*
Trans ID	Transaction record ID
Empid	Store employee id who issue the query
SqlQuery	The SQL query
Timestamp	Timestamp when the query occurs

4.2 Implementation of Two Testing IDS

We chose two insider-thread IDS to be tested on our benchmark – Role Based Access Control Classifier (RBAC classifier) proposed by Bertino, Kamr, Terzi and Vakali (2005), and Bayesian Network IDS proposed by An, Jutla and Cercone (2006). In the following section, we are going to explain how these two IDSs are being implemented in detail.

4.2.1 Implementation of RBAC Classifier

The procedure proposed by Lang (2005) is being used as a reference model to proceed in implementing the RBAC classifier. We chose Java 6.5 as our implementation platform for the RBAC classifier. Like all probabilistic classifiers, the RBAC classifier uses the Maximum Aposteriori Probability (MAP) decision rule; it arrives at the correct classification as long as the correct class is more probable than any other class. The nature of the naïve probability model enables the RBAC classifier to raise an alarm when the probability of a user acting according to the role he is claiming to have is low (Bertino, et al., 2005). In the classification problem, a set of training examples D_T is provided, and a new instance with attribute values $(a_1, ..., a_n)$ is given. The goal is to assign to this new instance the most probable class value v, given the attribute $(a_1, ... , a_n)$ that describes it:

$$
\begin{aligned}
v &= \max_{vj \in V} P(v_j \mid a_1, a_2 .... a_n) \\
&= \max_{vj \in V} \frac{P(a_1, a_2 .... a_n \mid v_j) P(v_j)}{P(a_1, a_2 .... a_n)} \\
&= \max_{vj \in V} P(a_1, a_2 .... a_n \mid v_j) P(v_j)
\end{aligned}
$$

$$
P(a_1, a_2 .... a_n \mid v_j) = \prod_i P(a_i \mid v_j)
$$

In order to build profiles, Bertino, et al. (2005) transforms the log file entries into a *triplet* format, which contain three fields (*SQL Command, Relation Information, Attribute Information*). For simplicity they further denoted the triplet as T(c, R, A), where c corresponds to the command, R to the relation information and A to the attribute information. According to the amount of information it stores, a triplet can classify as c-*triplet*, *m-triplet* and *f-triplet*, where c-triplet stores the least amount of information and f-triplet stores the most. By applying the above general Naïve Bayes Classifier model to RBAC classifier framework, Bertino, et al. (2005) defined the set of classifications is the set of roles r in the database system while the observations are the log-file triplets. Therefore, if R denotes the set of roles, predicting the role $r_j \in$ R of a given observation (c_i, R_i, A_i) requires, in accordance to the above naïve Bayesian equation:

$$r = \max_{rj \in R} P(r_j)P(c_i \mid r_j)P(R_i \mid r_j)P(A_i \mid r_j)$$

In our RBAC classifier implementation program, we took coarse triplet (c-triplet) and medium-grain triplet (m-triplet) observations into consideration. C-triplet (SQL-CMD, REL-COUNTER, ATTR-COUNTER) is the simplest log file entries format, which REL_COUNTER and ATTR_COUNTER contain the number of relations and attributes involved in the issued SQL query, respectively. The *SQL Command* field (SQL-CMD) of both c-triplet and m-triplet corresponds to the issued SQL command.

In m-triplet (SQL-CMD, REL-BIN[], ATTR-COUNTER[]), *Relation Information* field is a binary vector of size equal to the number of relations in the database, denoted as REL-BIN[]. Meanwhile, *Attribute Information* field of m-triplet is a vector, denoted as ATTR-COUNTER[], of size equal to the size of the REL-BIN[] vector. The *i*-th element of the ATTR-COUNTER[] vector corresponds to the number of attributes of the *i*-th relation that are involved in the *SQL command*. This bit vector contains 1 in its *i*-th position if the *i*-th relation is included in the SQL command. Table 6 shows two SQL commands corresponding to select statements and their representations according to the two triplets. In the example, they consider a database consisting of two relations R1= {A1, B1,C1} and R2={B2,D2,E2} (Bertino, et.al, 2005).

Table 6

Triplet construction example (Bertino, et.al, 2005)

SQL Command	c-triplet	m-triplet
Select A1,B1 From R1	Select <1> <2>	Select <1,0> <[1,1,0,0,0],[0,0,0,0,0]>
Select R1.A1, R1.C1, R2.B2, R2.D2 From R1,R2 Where R1.E1=R2.E1	Select <2> <4 >	Select <1,1> <[1,0,1,0,0],[0,1,0,1,0]>

We consider c-triplet as the default triplet format for RBAC classifier. M-triplet is also included in the implementation of RBAC classifier, so that the performance improvement with more information stored in the triplet can be tested.

There are three Java classes included*: RBAC classifier, NBlearner* and *DomainValue*. The *RBAC classifier class* is in charge of interacting with the benchmark. The *NBlearner class* and its methods implement the IDS algorithm. The *DomainValue class* is used to store attribute values. The class diagram is listed in Figure 4.8.

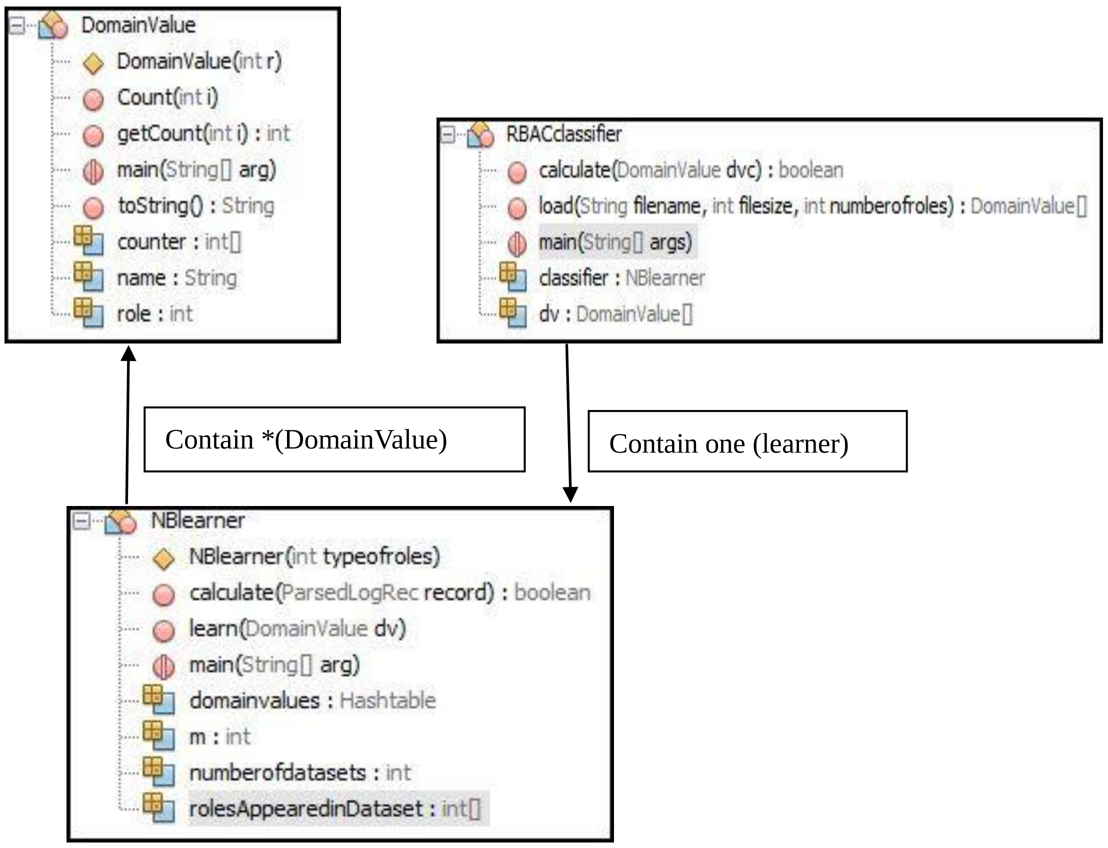


Figure 4.8 Class Diagram of RBAC Classifier

4.2.2 Implementation of Bayesian Network IDS (BNIDS)

The other insider threat IDS algorithm we chose was Bayesian Network IDS (BNIDS) proposed by An, et al., (2006). The core of our implementation is based on the application *GeNIe* and its Structural Modeling, Inference, and Learning Engine (jSMILE) for graphical probabilistic model of BNIDS, contributed to the community by the Decision Systems Laboratory of the University of Pittsburgh (http://dsl.sis.pitt.edu). *GeNIe* is a development environment for building graphical decision- theoretic models. As shown in Figure 4.9, BNIDS is built by using the GeNIe interface, where node *I* represents an insider intrusion while the others are observable events that may trigger an insider attack alarm, and the edges present conditional dependencies between nodes.

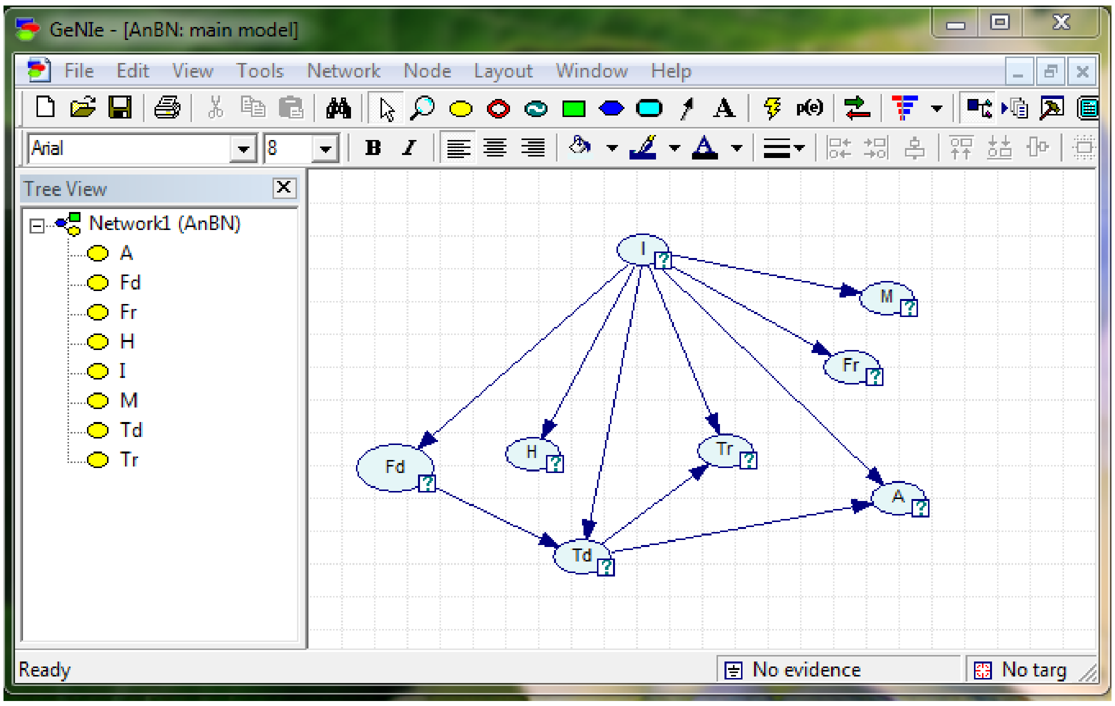

GeNIe - [AnBN: main model]
File Edit View Tools Network Node Layout Window Help
Tree View
Network1 (AnBN)
A
Fd
Fr
H
I
M
Td
Tr
I
M
Fr
Fd
H
Tr
A
Td
Ready
No evidence
No targ

Figure 4.9 BNIDS Diagram in Genie, where hypothesis variable *I* denotes a privacy intrusion. All others are information variables, where *Fd* denotes the usage frequency of the database, *H* office hours, *Td* time spend on database, *Tr* time spent on special records, *A* the amount of records accessed, *M* the number of special records modified, and *Fr* the usage frequency of records. Conditional probability distributions between nodes and prior knowledge about *Tr* and *A*, are assigned as suggested by An, et al. (2006).

In our benchmark, event *A* is monitored by @@rowcount which we retrieved from queries; *H* is monitored by the timestamp of each query ; *Fr* and *Fd* are observed by frequency; *Td* by the time span between login and logout; *Tr* is observed by the time difference between current query and the previous one issued by the same employee. *M* is monitored by the number of records that have been modified by the UPDATE or DELETE command; we considered that all the records stored in the database are sensitive.

Chapter 5

Experiment Methodology and Results

In this chapter, the performances of two IDSs are tested with four test cases. The IDSs run against the benchmark with varying percentage of anomalies, workload, number of roles, and attack patterns individually. We also demonstrate the usefulness of benchmark to tune an IDS setting.

5.1 Scalability Testing

In our benchmark, we provide configurable environments to conduct various performance tests. First, we want to test the accuracy and effectiveness of both IDSs in a small financial bank scenario. Test case 1(*Figure* 5.1 and *Figure* 5.2) established a testing environment with increasing number of injected anomalies, and constant training data (200 records) and normal background data of 1000 records. In Test Case 1, we assume there are two branches and 25 employees working for each of the branch. In other words, the bank has 50 employees, each of which belongs to one of the 20 roles we used in Test Case 1. The threshold's default value for BNIDS is 0.5. Later in section 5.3, we will demonstrate the procedure of using benchmark to tune the threshold setting and how it will affect the IDS performance.

Figure 5.1 shows a superior performance in precision of Bayesian Network Intrusion Detection System (BNIDS), proposed by An, Jutla and Cercone (2006), compared to the Role-Based Access Control Classifier (RBAC Classifier), proposed by Bertino, et al. (2005). In the test case 1, both IDSs show, given the constant size of normal background, the more anomalies that are injected in the testing set, the higher the precision. Anomalies are drawn from the anomaly templates under uniform distribution; each of the anomalies belongs to one of the three insider attacks: *Data Harvest, Masquerade* and *Sabotage*.

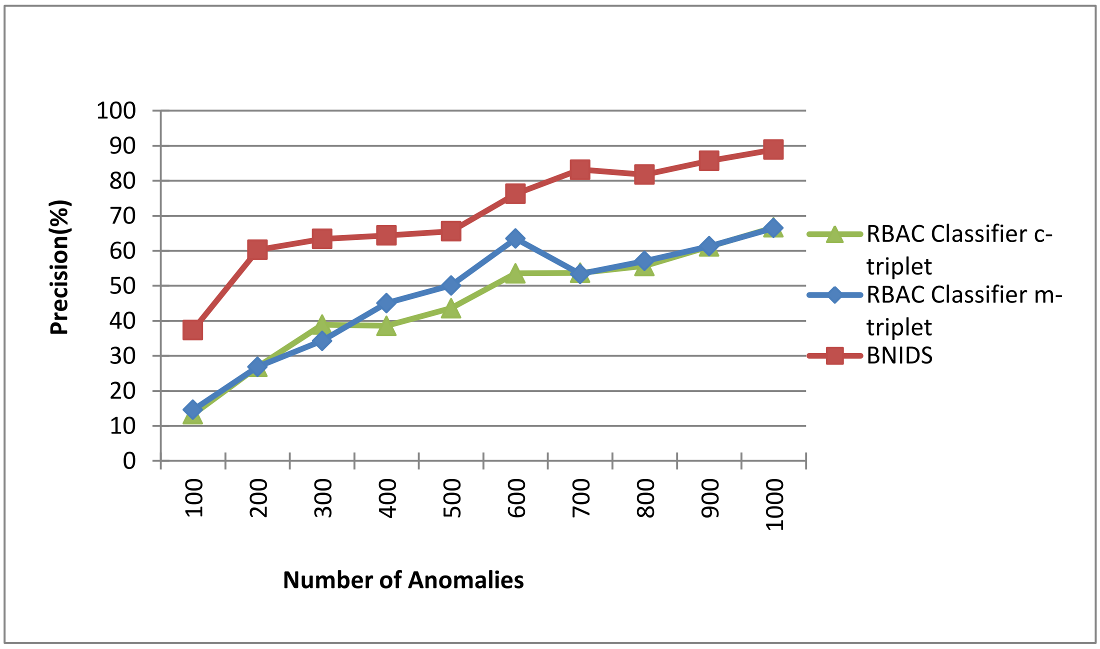


Figure 5.1 Test Case 1: Precision vs. Number of Injected Anomalies

Our RBAC Classifier uses a Naïve Bayes classifier. It assumes that all attributes are statistically independent. In other words, all attributes should have the same relevance with respect to the classification task. The reason for making this assumption can be found in the way Naïve Bayes classifier computes the class likelihoods. They are simply products of the conditional probabilities of all attributes values given the class value. In our case, attributes are represented by various observable database events, while classes are represented by all the employee roles existing in the database. On the other hand, the Bayesian network structure used by BNIDS captures that some attributes are in fact dependent.

As shown in Figure 5.1, we received an improvement in precision when the percentage of anomalies in the workload goes up. It shows that the Bayesian Network inference makes use of dependencies, thus outperforms Naïve Bayes classifier by 20% on precision. The latter cannot handle dependent attributes and therefore needs the attribute independence assumption and end up with a lower performance in precision. Moreover, the RBAC Classifier only considers data manipulation as observable events, while BNIDS includes database events like usage frequency of database (*Fd*) and office hour (*H*) to capture the context of an intrusion threat scenario better. For this reason, a lower number of false positives is expected from BNIDS than RBAC classifier given the same testing environment. Hence, a higher precision is gained by BNIDS over RBAC classifier.

In Figure 5.1, we also witnessed that m-triplet RBAC classifier, with its larger information stored in the triplet, shows a slightly higher performance than the c-triplet RBAC implementation.

Figure 5.2 shows a steady performance in recall around 82% for BNIDS and 95% for RBAC classifier. RBAC classifier outperforms BNIDS by 10% in terms of recall. In our implementation for BNIDS, each role uses the same conditional probability distributions between nodes and prior knowledge as suggested by An, et al.

(2006), which results in a higher number of false negatives. Therefore, we got a lower recall for BNIDS. In test case 1, we maintained the threshold of BNIDS at 0.5. That means if the probability of occurrence of intrusion is larger than 50%, the IDS consider it is an intrusion attack. In order to achieve a higher recall, we can lower the threshold to allow more true positives are being caught by IDS. However, with a lower threshold, more false positives are being detected by BNIDS as well, which may contribute to a lower precision. We will discuss how to use the benchmark to find an ideal balance between precision and recall later in section 5.3.

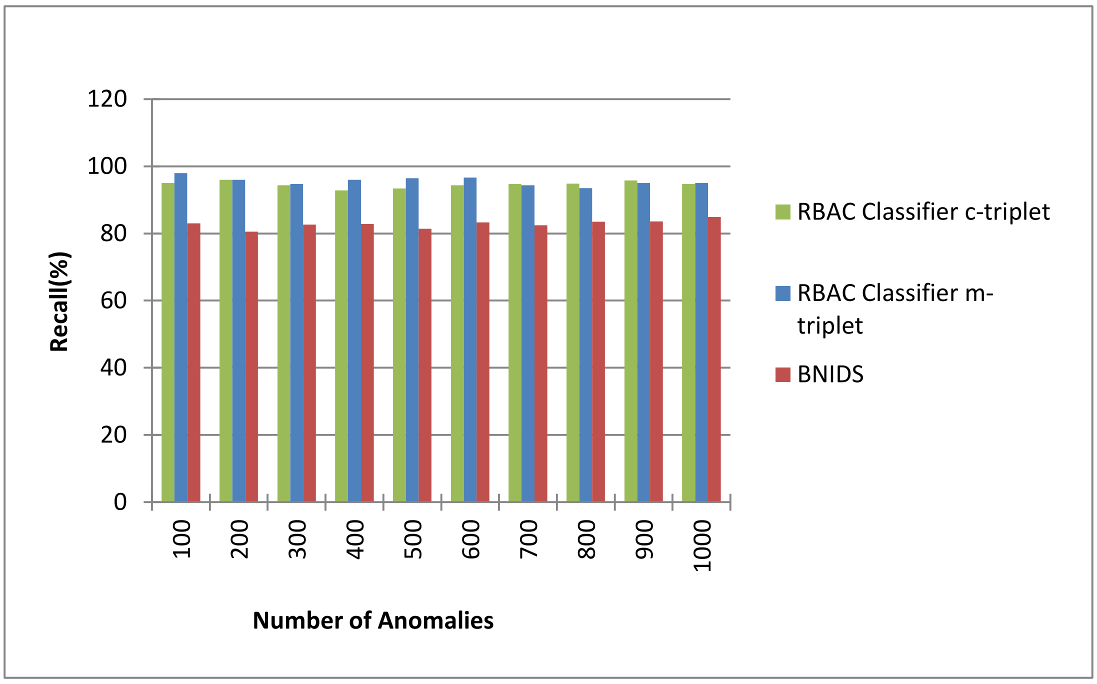


Figure 5.2 Test Case 1: Recall vs. Number of Injected Anomalies

According to the results we gathered from test case 1, we now can calculate the F_β-measure where β =1. F-measure is the harmonic mean of precision and recall.

With steady performances in recall and growing precisions, the F-measure values for both IDSs increase as the percentage of anomalies in the workload goes up. As shown in figure 5.3, the overall performance of BNIDS increases from 52% to around 87% when there are more anomalies injected in the workload. Meanwhile, RBAC Classifier shows noticeable improvement in overall performance before 600-injected-anomalies mark, and it remains stable around 75% afterwards.

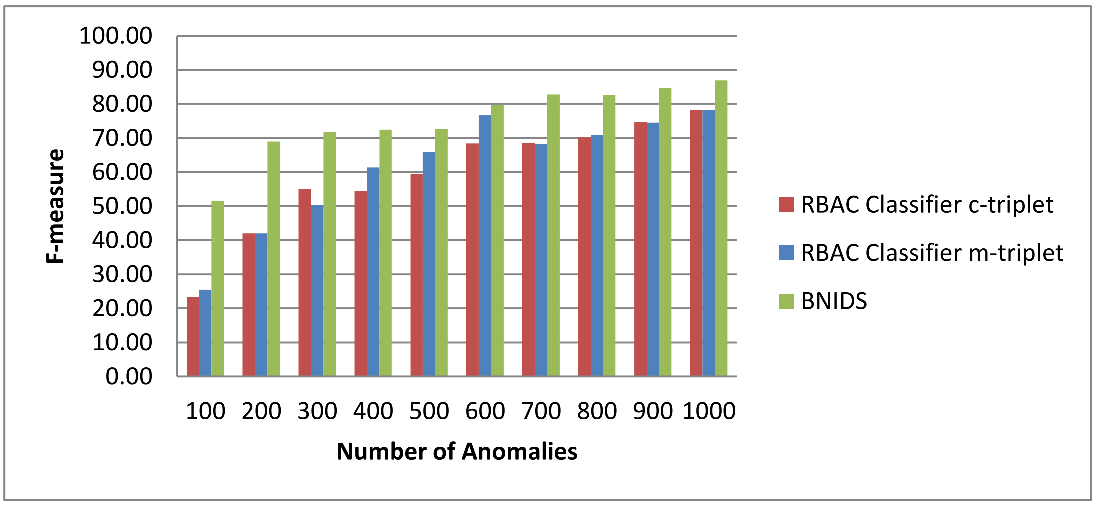


Figure 5.3 Test Case 1: F-measure vs. Number of Anomalies

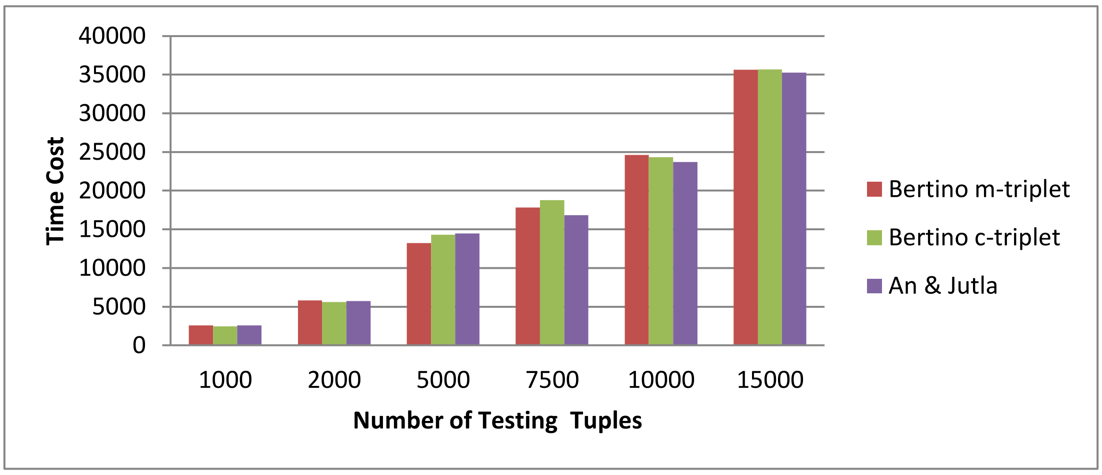


Figure 5.4. Test Case 1: Time Cost vs. Number of Testing Tuples

Next, we built an experimental environment that monitors the time cost for running the IDSs with increasing total number of testing tuples. The results are demonstrated in Figure 5.4, we can tell that with larger log to process, the time cost went up. However, there were not many differences among IDSs in terms of time spent on inference. It needs to be mentioned that we did not include the training phase of both IDSs, considering an IDS only needs to be trained once per test case, and BNIDS already provides reasonable conditional probabilities for its Bayesian network structure (An & Jutla, 2006).

In test case 2, we want to check the performance of both IDSs in a larger financial bank and where the percentage of anomalies that exist in the workload is low. In this test case we establish a large financial bank scenario, where the bank has around 10,000 customers across 10 branches, each of which has 50 employees. In other words, the total number of employees is around 500. In this test case, we use 20 roles in the workload. During the test, we maintain the percentage of anomalies at 10% while increasing the total number of workload records.

As expected, as shown in Figure 5.5, BNIDS outperformed RBAC Classifier in terms of precision. Both IDSs show low precision compared to other test cases. However, the result is reasonable considering there are only 10% of anomalies within the workload; with higher total number of false positives and lower total number of true positives, a lower precision is expected.

In Figure 5.6, both IDSs show a steady performance in terms of recall around

85% for BNIDS and 95% for RBAC classifier. That means most of the anomalies are detected by both IDSs despite different sizes of the workload.

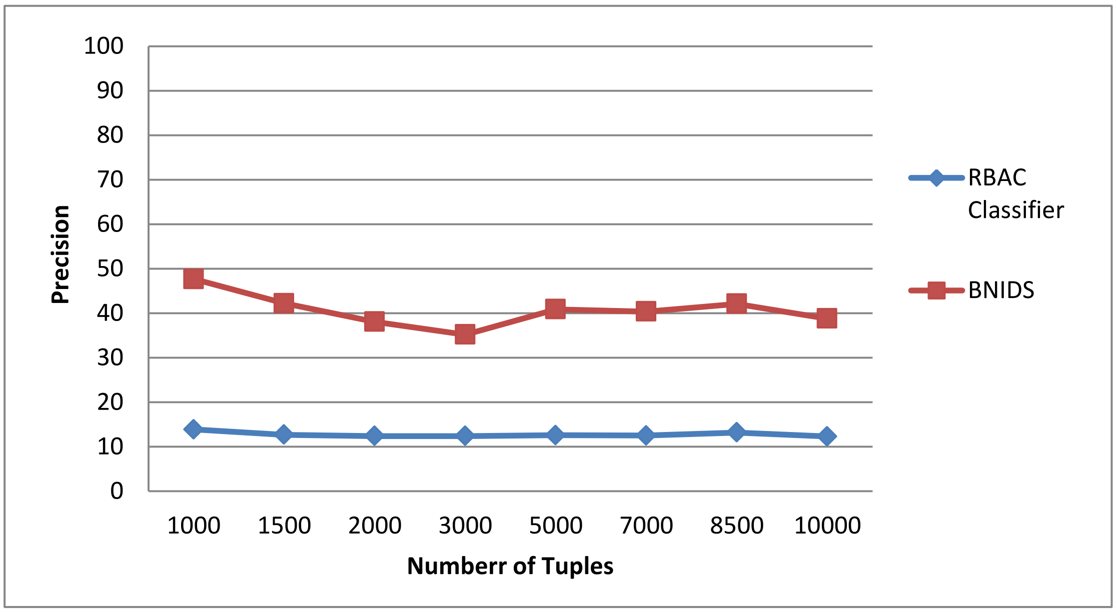


Figure 5.5 Test Case 2: Precision vs. Number of Tuples

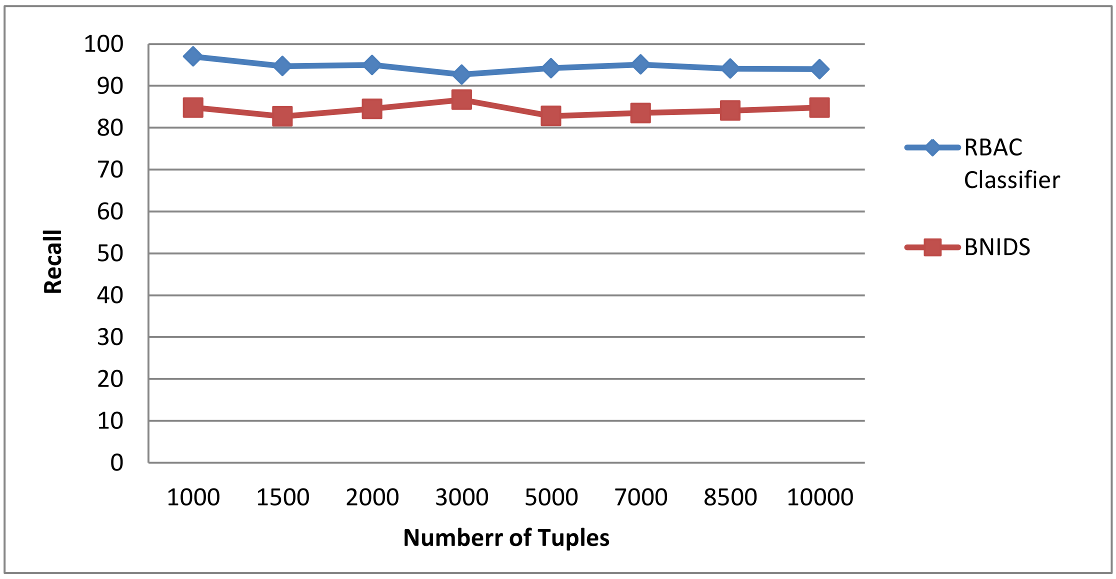


Figure 5.6 Test Case 2: Recall vs. Number of Tuples

In previous two test cases, there are 20 roles which exist in the workload. We are wondering whether the total number of roles will affect the accuracy of the performance. With that in mind, we established test case 3 as a testing environment with varying number of roles, and constant training data (200 tuples) and testing data, which has 1000 tuples of normal background and 100 injected anomalies. In Figure 5.7, RBAC classifier shows a steady performance of precision around 15%, despite various total numbers of roles in the workload. BNIDS out performed RBAC by 20% when there are less than12 roles in the workload; and increase the gap to 35% when there are more than 12 roles in the workload. The size of the training data set was varied from 100 to 1000 tuples with no difference in results. Thus a size of 200 tuples for the raining data is deemed adequate.

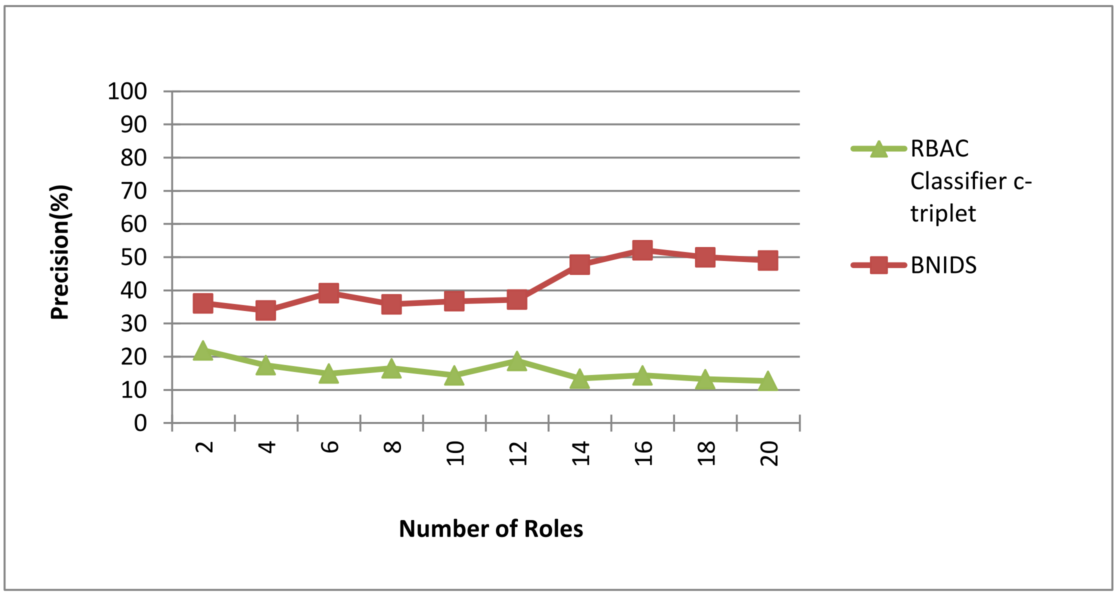


Figure 5.7 Test Case 3: Precision vs. Number of Roles

In Figure 5.8, BNIDS shows steady performance in terms of recall while RBAC Classifier has increasing recall with higher total number of roles in the workload. In other words, RBAC Classifier has better classification results when there are more roles (classes) to work on.

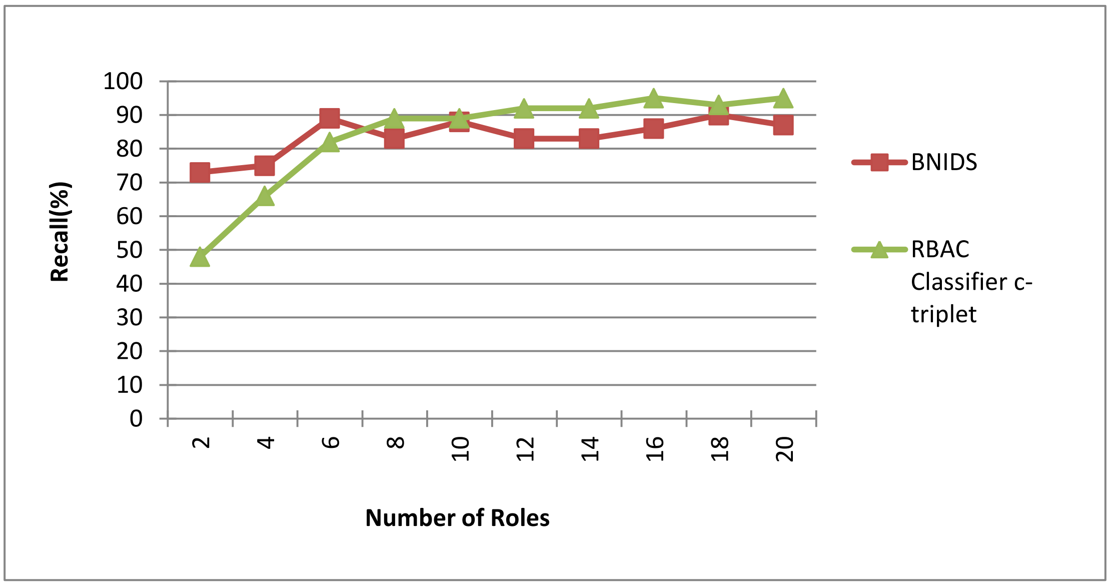


Figure 5.8 Test Case 3: Recall vs. Number of Roles

5.2 Different Anomaly Injection Methods Testing

Test case 4 is a variant of Test case 1 with different anomalies injection methods: *Gaussian Injection* and *Uniform Injection*. In this test case, we want to see whether attack patterns will affect IDSs results. As shown in Figure 5.9 and Figure 5.10, the RBAC classifier did not show any performance difference between the two anomaly injection methods.

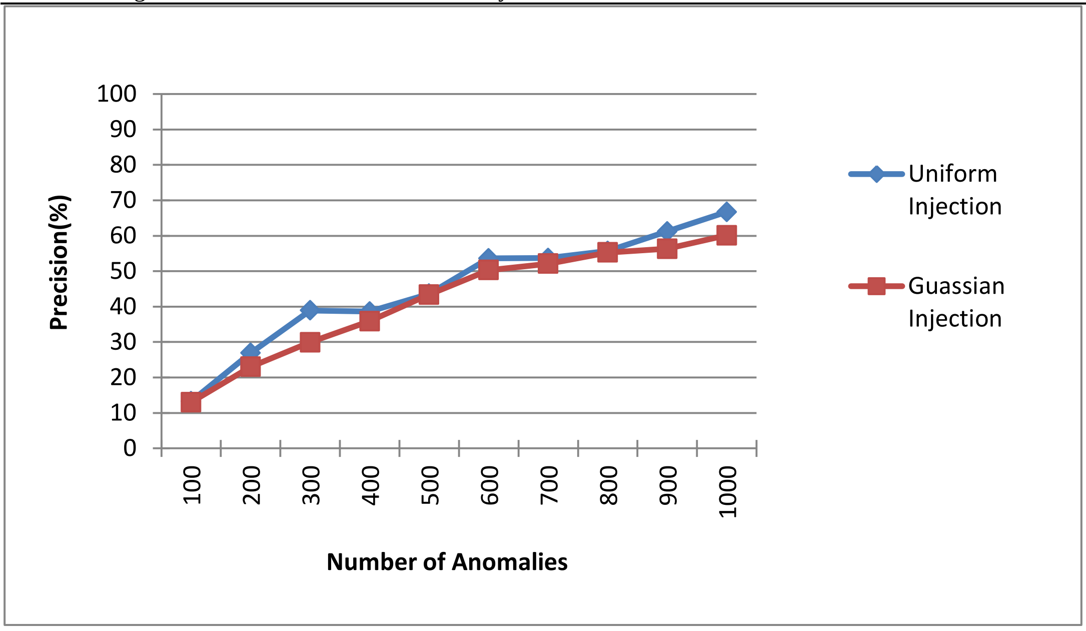


Figure 5.9 Test Case 4: Precision vs. Number of Anomalies RBAC-Classifier with c-triplet

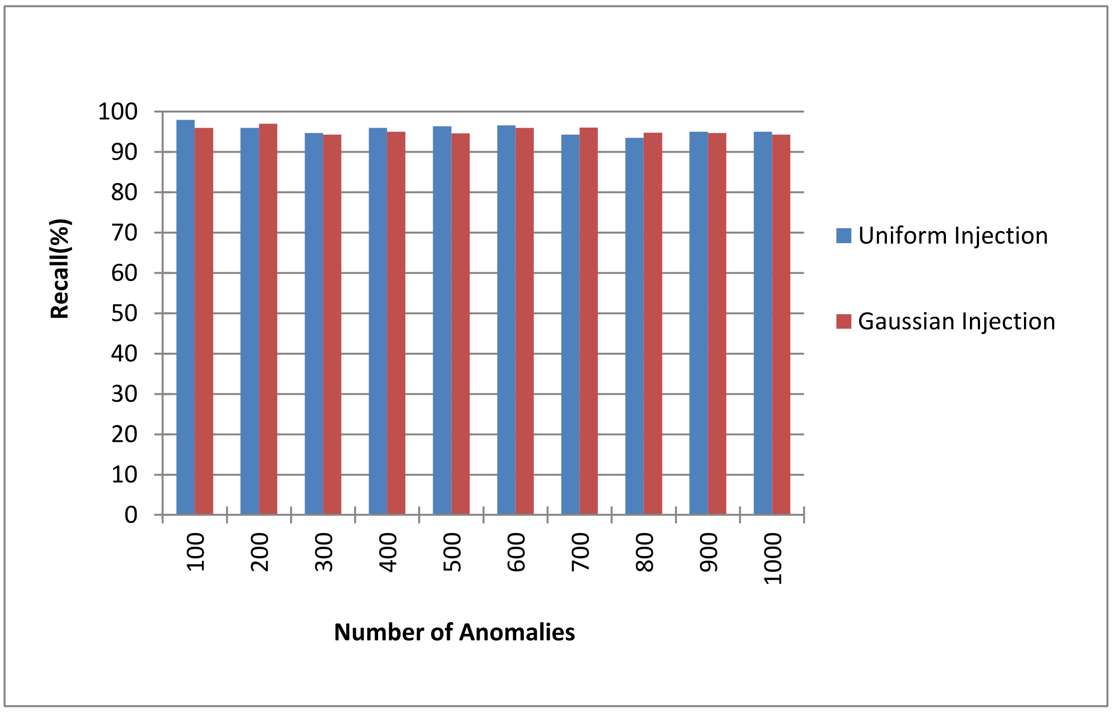


Figure 5.10 Test Case 4: Recall vs. Number of Anomalies RBAC-Classifier with c-triplet

Meanwhile, as shown in Figure 5.11 BNIDS has a lower precision before

300-injected-anomalies with Gaussian distribution. Figure 5.12 demonstrates similar results in recall for both injection methods. Gaussian distribution is a continuous distribution. When the number of anomalies is less than 300, the distribution is not smooth enough to assimilate the real Gaussian distribution. When the number of anomalies is greater than 300, the distribution in the experiment is smooth enough to be similar to the real Gaussian distribution.

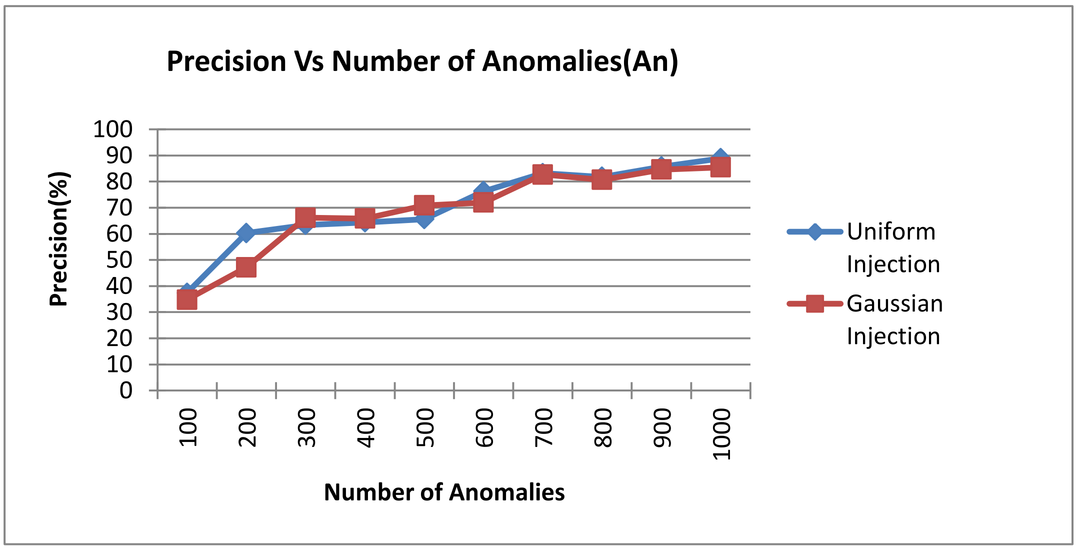


Figure 5.11 Test Case 4: Precision vs. Number of Anomalies with BNIDS

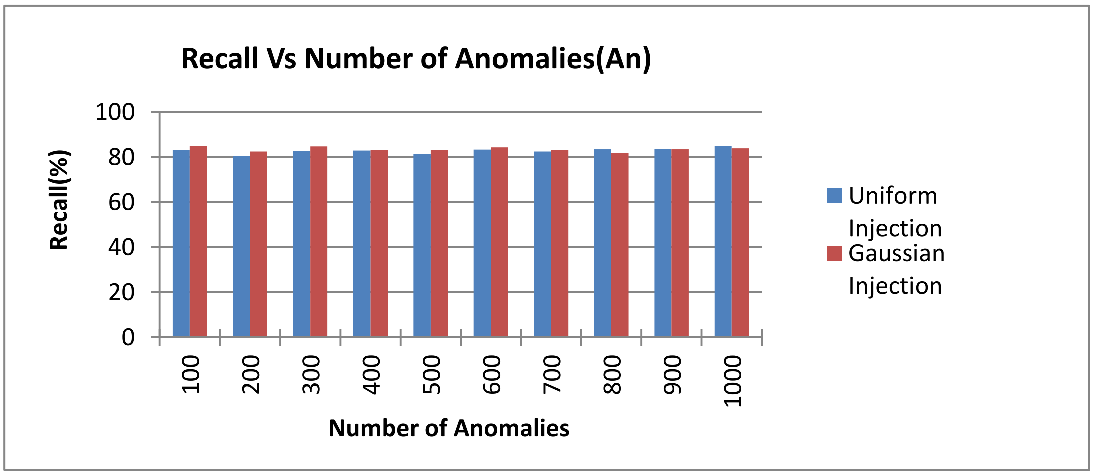


Figure 5.12 Test Case 4: Recall vs. Number of Anomalies with BNIDS

5.3 Tuning IDS Settings with Benchmark

For an IDS to work effectively, it should be tuned correctly. Settings of the IDS need to be in compliance with the security policies and goals of the sytem administrator. Although, the tuning process is very time consuming, it is a very vital factor to an effective IDS configuration. Later we will use BNIDS as an example to demonstrate how to utilize our benchmark for tuning IDSs settings.

Here, we set up a test financial bank scenario that has two branches, each of which has 25 employees. The workload has constant 1000 tuples with 10% anomalies. By raising the threshold of BNIDS from 0.1 to 1.0 with the increment of 0.1, we received 10 sets of results of precision and recall, which are shown in Figure 5.13.

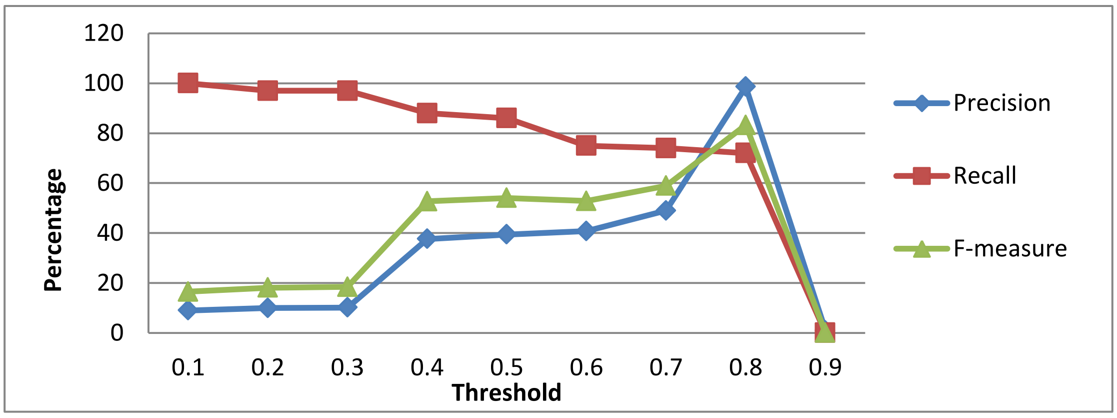


Figure 5.13 BNIDS Threshold Tuning

Figure 5.13 shows a gradually dropping recall numbers and increasing precision percentages when the threshold goes higher. When the threshold reach at 1, both recall and precision dropped to 0, where there are no true positives. With lower threshold, more true positives are being caught by IDS, which contributes to a higher recall; meanwhile, more normal events are recognized as anomalies, hence more false positives and a lower precision. Increasing the threshold can decrease the number of normal events being wrongly classified as anomalies; hence less false positives and a higher precision.

The goal is to find a ideal balance between precision and recall. Thresholds need to be tuned to ensure the IDS could identify relevant data without overloading the administrator with warnings or too many false positives. The F-measure is used as an overall performance metric in our benchmark. As shown in Figure 5.13, the F-measure reach its peak value at 0.8. With configured threshold of 0.8, we followed the procedure in test case 1 by maintaining normal background at 10000 records and increasing the number of anomalies in the workload. Figures 5.14 and 5.15 show a much better performance in terms of precision of the BNIDS over the RBAC Classifiers, however, the latter out perform by 10% in terms of the recall.

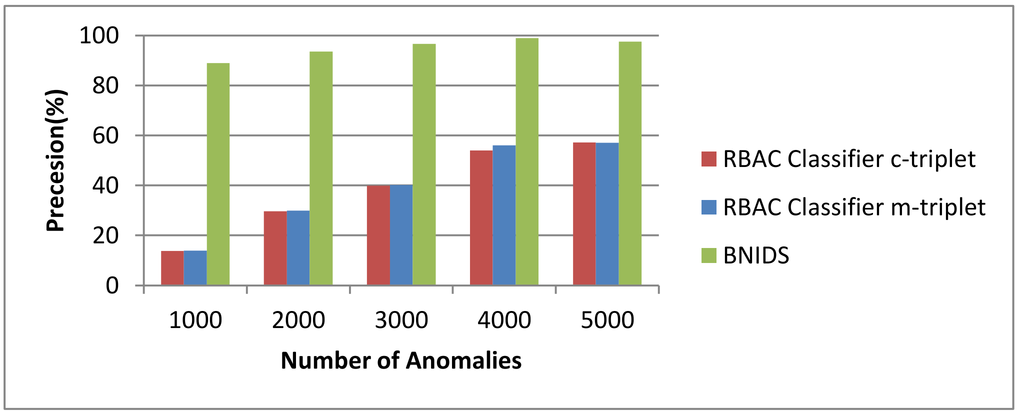


Figure 5.14 Precision vs. Number of Injected Anomalies

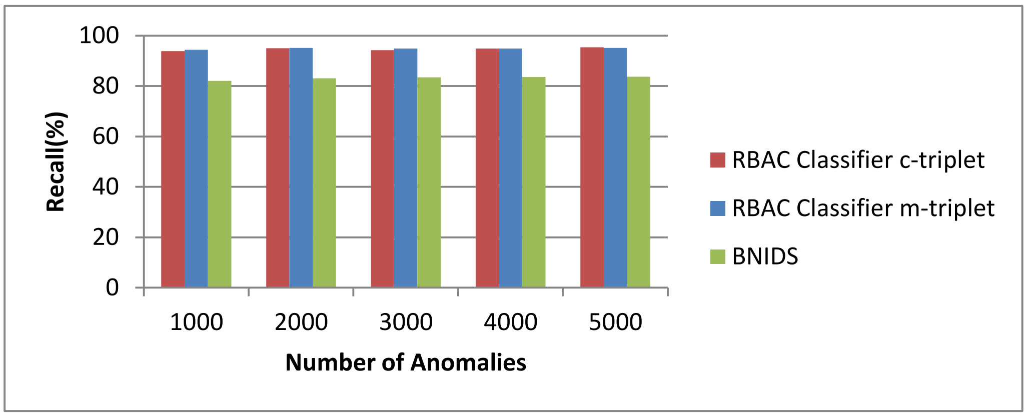


Figure 5.15 Recall vs. Number of Injected Anomalies

Chapter 6

Conclusion and Future Work

6.1 Conclusion

We have presented a benchmark for insider threat intrusion detection systems, and have tested two representitative insider threat intrusion detection algorithms with this benchmark. The previous chapters demonstrated the usefulness of our insider threat benchmark by comparing the performances of different intrusion detection systems across various testing environments.

Little work has been done to provide a performance platform for insider threat IDSs to this date. This study contains original research to create a benchmark to compare different Intrusion Detection algorithms through the development of a synthetic workload that mirrors the types of insider attacks that exist in today’s database environment.

The foremost work on a benchmark’s agenda is to ascertain the limits to scalability we are witnessing with the workload generator as well as understanding the performances, i.e reall and precision, of the IDSs to be compared. With severl test cases conducted in this paper, we have witnessed a slightly better performance in recall for RBAC classifier over BNIDS. However, BNIDS showed a much better performance in precision over RBAC classifier, especially with tuned thresholds.

By running IDSs under various test senarios, the benchmark allows system administrators to make informed decisions to select an appropriate IDS, according to provided performance metrics, to best identify insider attacks and abuse against information systems for their environment.

6.2 Future Work

Future work includes creating benchmark workloads for other contexts, e.g. health care industry, R&D companies, and government agencies. When implementing the benchmark, users should focus on which information the organization wants the IDS to protect and which group of people could get access to the data. Users can reference section 3.2 to select proper benchmark parameters and performance metrics. In our benchmark, we provide three composite metrics - precision, recall and F-measure.

It would also be useful to look at enriching the metric set, such as receiver operating characteristic (ROC) curve, the area under the curve (AUC), and execution time/cost. By plotting the fraction of TPR vs FPR, the ROC curve can be created. It shows the probablitiy of detection provided by the IDS at a given false alarm rate. AUC can be used to compare the testing IDSs, the larger the area under the curve, the better of the performance. In our benchmark, we already provided the option to monitor the execution time for testing the IDSs. By assigning a cost to each query plan, execution time/cost can be calculated.

References

Agrawal, R., Bird, P., Grandison, T., Kiernan, J., Logan, S., & Rjaibi, W. (2005). Extending relational database systems to automatically enforce privacy policies. *Proceedings of the 21st International Conference on Data Engineering,* 1013-1022. doi: 10.1109/ICDE.2005.64

Agrawal, R., Kiernan, J., Srikant, R., & Xu, Y. (2002). Hippocratic databases. *Proceedings of the 28th International Conference on very Large Data Bases,* Hong Kong, China. 143-154.

Aleman-Meza, B., Burns, P., Eavenson, M., Palaniswami, D., & Sheth, A. (2005). An ontological approach to the document access problem of insider threat. *Intelligence and Security Informatics*, 45-47.

An, X., Jutla, D., & Cercone, N. (2006). A bayesian network approach to detecting privacy intrusion. *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology,* 73-76. doi: 10.1109/WI-IATW.2006.6

Anderson, J. P. (1980). *Computer security threat monitoring and surveillance* (Vol. 17). Technical report, James P. Anderson Company, Fort Washington, Pennsylvania.

Anton, A. I., Bertino, E., Li, N., & Yu, T. (2007). A roadmap for comprehensive online privacy policy management. *Commun.ACM, 50*(7), 109-116. doi: 10.1145/1272516.1272522

Athanasiades, N., Abler, R., Levine, J., Owen, H., & Riley, G. (2003). Intrusion detection testing and benchmarking methodologies. *Information Assurance, 2003. IWIAS 2003. Proceedings. First IEEE International Workshop on,* 63-72.

Axelsson, S. (1999). The base-rate fallacy and its implications for the difficulty of intrusion detection. *Proceedings of the 6th ACM Conference on Computer and Communications Security,* Kent Ridge Digital Labs, Singapore. 1-7. doi: 10.1145/319709.319710

Bank Transactions.

http://www.turtlesoft.com/Goldenseal-Software-Reference/banktran.htm#transact_type

(ACSAC'05), Tucson, Arizona 5-9 Dec. 2005.

Bertino, E., Kamra, A., Terzi, E., & Vakali, A. (2005). Intrusion detection in RBAC-administered databases. *Proceedings of the 21st Annual Computer Security Applications Conference,* 170-182. doi: 10.1109/CSAC.2005.33

Bodlaender, M. P., Stok, P. D. V. v. d., & Son, S. H. (1997). A transaction-based temporal data model for real-time databases. *Proceedings of the 1997 Joint Workshop on Parallel and Distributed Real-Time Systems (WPDRTS / OORTS '97),* 149.

Calzarossa, M., & Serazzi, G. (1993). Workload characterization: A survey. *Proceedings of the IEEE, 81*(8), 1136-1150.

Cert Research Report (2010)

Retrieved from http://www.cert.org/research/researchreport.html

Chinchor, N., MUC-4 Evaluation Merics, *in Proc. of the Forth Message Understanding Conference*, pp 22-29, 1992.

Chung, C. Y., Gertz, M., & Levitt, K. (2000). Demids: A misuse detection system for database systems. *Third International IFIP TC-11 WG11*, *5*, 159-178.

Conte, T. M., & Hwu, W. W. (1991). Benchmark characterization. *Computer, 24*(1), 48-56. doi: 10.1109/2.67193

Denning, D. E., Lunt, T. F., Schell, R. R., Shockley, W. R., & Heckman, M. (1988). The SeaView security model. *Security and Privacy, 1988. Proceedings., 1988 IEEE Symposium on,* 218-233.

Didriksen, T. (1997). Rule based database access control—a practical approach. *Proceedings of the Second ACM Workshop on Role-Based Access Control,* Fairfax, Virginia, United States. 143-151. doi: 10.1145/266741.266772

Dietrich, S. W., Brown, M., Cortes-Rello, E., & Wunderlin, S. (1992). A practitioner's introduction to database performance benchmarks and measurements. *Comput.J., 35*(4), 322-331. doi: 10.1093/comjnl/35.4.322

Forrest, S., Hofmeyr, S. A., & Somayaji, A. (1997). Computer immunology. *Commun.ACM, 40*(10), 88-96. doi: 10.1145/262793.262811

Freedmen, R., Maurer, J., Wolfe, V., Wohlever, S., Milligan, M., & Thuraisingham, B. (2000). Benchmarking real-time distributed object management systems for evolvable and adaptable command and control applications. In *Object-Oriented Real-Time Distributed Computing, 2000.(ISORC 2000) Proceedings. Third IEEE International Symposium on* (pp. 202-205). IEEE.

Gaffney, J. E., Jr., & Ulvila, J. W. (2001). Evaluation of intrusion detectors: A decision theory approach. *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on,* 50-61.

Geppert, A., Gatziu, S., & Dittrich, K. R. (1995). A designer's benchmark for active database management systems: Oo7 meets the BEAST. *Proceedings of the Second International Workshop on Rules in Database Systems,* 309-326.

Gu, G., Fogla, P., Dagon, D., Lee, W., & Skoric Boris. (2006). Measuring intrusion detection capability: An information-theoretic approach. *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security,* Taipei, Taiwan. 90-101. doi: 10.1145/1128817.1128834

Huang, X., Peng, F., An, A., & Schuurmans, D. (2004). Dynamic web log session identification with statistical language models. *J.Am.Soc.Inf.Sci.Technol., 55*(14), 1290-1303. doi: 10.1002/asi.20084

Hunker, J., & Probst, C. W. (2011). Insiders and insider threats—an overview of definitions and mitigation techniques. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, *2*(1), 4-27.

Hopcroft, J., & Ullman, J. (2008). Introduction to Automata Theory Languages and Computation., 3rd Edition, Pearson Education.

Jain, R. (1991). *The art of computer systems performance analysis* (Vol. 182). New York: John Wiley & Sons.

Kamel, N., & King, R. (1992). Intelligent database caching through the use of page-answers and page-traces. *ACM Trans.Database Syst., 17*(4), 601-646. doi: 10.1145/146931.146933

Karjoth, G. (2003). Access control with IBM tivoli access manager. *ACM Trans.Inf.Syst.Secur., 6*(2), 232-257. doi: 10.1145/762476.762479

Kayacik, H. G., & Zincir-Heywood, N. (2005). Analysis of three intrusion detection system benchmark datasets using machine learning algorithms. *Proceedings of the 2005 IEEE International Conference on Intelligence and Security Informatics,* Atlanta, GA. 362-367. doi: 10.1007/11427995_29

Kearns, J. P., & DeFazio, S. (1989, April). Diversity in database reference behavior. In *ACM SIGMETRICS Performance Evaluation Review* (Vol. 17, No. 1, pp. 11-19). ACM.

Lane, T., & Brodley, C. E. (1999). Temporal sequence learning and data reduction for anomaly detection. *ACM Trans.Inf.Syst.Secur., 2*(3), 295-331. doi: 10.1145/322510.322526

Lang, M. (2002). Implementation of Naive Bayesian Classifiers in Java. In *Proceedings of the Conference on Intelligent Systems and Intelligent Agents*.

Lee, V. C. S., Stankovic, J. A., & Son, S. H. (2000). Intrusion detection in real-time database systems via time signatures. *Proceedings of the Sixth IEEE Real Time Technology and Applications Symposium (RTAS 2000),* 124.

Lundin, E., & Jonsson, E. (2000). Anomaly-based intrusion detection: Privacy concerns and other problems. *Comput.Netw., 34*(4), 623-640. doi: 10.1016/S1389-1286(00)00134-1

Lunt, T. F., Tamaru, A., Gilham, F., Jagannathan, R., Neumann, P. G., & Jalali, C. (1990). IDES: A progress report [intrusion-detection expert system]. *Computer Security Applications Conference, 1990., Proceedings of the Sixth Annual,* 273-285.

Maybury, M., Chase, P., Cheikes, B., Brackney, D., Matzner, S., Hetherington, T., ... & Longstaff, T. (2005). *Analysis and detection of malicious insiders*. MITRE CORP BEDFORD MA.

Maxion, R. A., & Tan, K. M. C. (2000). Benchmarking anomaly-based detection systems. *Dependable Systems and Networks, 2000. DSN 2000. Proceedings International Conference on,* 623-630.

Moore, A. P., Capelli, D. M., Caron, T. C., Shaw, E., Spooner, D., & Trzeciak, R. F. (2011). *A preliminary model of insider theft of intellectual property*. CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST.

Mun, H., Han, K., Yeun, C., & Kim, K. (2008). Yet another intrusion detection system against insider attacks. *Proc. of SCIS 2008*.

Peng Liu. (2002a). Architectures for intrusion tolerant database systems. *Computer Security Applications Conference, 2002. Proceedings. 18th Annual,* 311-320.

Peng Liu. (2002b). Architectures for intrusion tolerant database systems. *Computer Security Applications Conference, 2002. Proceedings. 18th Annual,* 311-320.

Ponder, C. (1990). Performance variation across benchmark suites. *SIGMETRICS Perform.Eval.Rev., 18*(3), 42-48. doi: 10.1145/122235.122238

Qian, J., Xu, C., & Shi, M. (2006). Redesign and implementation of evaluation dataset for intrusion detection system. *Proceedings of the 2006 International Conference on Emerging Trends in Information and Communication Security,* Freiburg, Germany. 451-465. doi: 10.1007/11766155_32

Qian, X., Stickel, M. E., Karp, P. D., Lunt, T. F., & Garvey, T. D. (1993). Detection and elimination of inference channels in multilevel relational database systems. *Research in Security and Privacy, 1993. Proceedings., 1993 IEEE Computer Society Symposium on,* 196-205.

Saavedra, R. H., & Smith, A. J. (1996). Analysis of benchmark characteristics and benchmark performance prediction. *ACM Trans.Comput.Syst., 14*(4), 344-384. doi: 10.1145/235543.235545

Sacco, G. M., & Schkolnick, M. (1986). Buffer management in relational database systems. *ACM Transactions on Database Systems (TODS)*, *11*(4), 473-498.

Salem, M. B., Hershkop, S., & Stolfo, S. J. (2008). A survey of insider attack detection research. *Insider Attack and Cyber Security*, 69-90.

Shirdhonkar, M. S., & Kokare, M. B. (2011). Document image retrieval using signature as query. *Computer and Communication Technology (ICCCT), 2011 2nd International Conference on,* 66-70.

Schultz, E. E. (2002). A framework for understanding and predicting insider attacks. *Computers & Security*, *21*(6), 526-531.

Sinha, S., Jahanian, F., & Patel, J. M. (2006). WIND: Workload-aware INtrusion detection. *Proceedings of the 9th International Conference on Recent Advances in Intrusion Detection,* Hamburg, Germany. 290-310. doi: 10.1007/11856214_15

Srikant, R., & Agrawal, R. (1996). Mining sequential patterns: Generalizations and performance improvements. *Proceedings of the 5th International Conference on Extending Database Technology: Advances in Database Technology,* 3-17.

Teng, H. S., Chen, K., & Lu, S. C. (1990). Adaptive real-time anomaly detection using inductively generated sequential patterns. *Research in Security and Privacy, 1990. Proceedings., 1990 IEEE Computer Society Symposium on,* 278-284.

Van Rijsbergen, C.J.(1979).Information Retrieval(1nd edition), Butterworth, London.

Venter, H. S., Olivier, M. S., & Eloff, J. H. (2004). PIDS: a privacy intrusion detection system. *Internet Research*, *14*(5), 360-365.

Yao, Q., An, A., & Huang, X. (2005). Finding and analyzing database user sessions. *Proceedings of the 10th International Conference on Database Systems for Advanced Applications,* Beijing, China. 851-862. doi: 10.1007/11408079_77

Yao, S. B., Hevner, A. R., & Young-Myers, H. (1987). Analysis of database system architectures using benchmarks. *Software Engineering, IEEE Transactions on, SE-13*(6), 709-725.

Yi Hu, & Panda, B. (2003). Identification of malicious transactions in database systems. *Database Engineering and Applications Symposium, 2003. Proceedings. Seventh International,* 329-335.

Zeng-Chang Qin. (2005). ROC analysis for predictions made by probabilistic classifiers. *Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on, , 5* 3119-3124 Vol. 5.

Zhang, X., Lu, X., Shi, Q., Xu, X. Q., Leung, H. C. E., Harris, L. N., ... & Wong, W. H. (2006). Recursive SVM feature selection and sample classification for mass-spectrometry and microarray data. *BMC bioinformatics*, *7*(1), 197.