CHAPTER – FOUR

Agent 1: The Biometric Template Storage Intrusion Detection Assistant

4.1 Introduction

4.2 Biometric storage
   4.2.1. Biometric Template
   4.2.2. Available biometric template storage
   4.2.3. Vulnerabilities in a Biometric Template Storage

4.3 Auditing used for Intrusion detection
   4.3.1. Overview of audit concept
   4.3.2. Auditing : Tool for Intrusion detection
   4.3.3. Auditing using RDBMS

4.4 Proposed System
   4.4.1. Architecture of proposed system
   4.4.2. Logic used to develop the proposed system
   4.4.3. Back tracing used for source detection
   4.4.4. Encode the Rules used for this agent

4.5 Findings of Agent 1

4.6 Prevention Technique suggested

Concluding Remark
PREVIEW

This Chapter is divided into four primary sections. The first section provides an overview of biometric template, biometric template storage and vulnerabilities in biometric storage. The second section describes audit concept, auditing used in intrusion detection system and how auditing can be done with Oracle. The third section provides the architecture of proposed intelligent agent which can be implemented on biometric template storage, logic used to develop this Knowledge based agent and rules used to detect intrusion. Fourth section explains output of this proposed agent which acts as intelligent assistant tool for security administrator.

4.1 Introduction

A biometric system is essentially a pattern recognition system that operates by capturing a biometric image from an individual, extracting a unique feature set from the acquired data, and comparing this feature set against the template set in the database. A system database consisting of biometric templates must be created through a process of enrolment. The user then comes back to the same system that he/she has enrolled on and tries to authenticate through that stored template. The extracted components are run through an algorithm and stored as a template.

The biometric authentication systems are used either in centralized or distributed architecture. They mostly differ by how the processing steps for biometric authentication system are divided between different machines. The attacks on stored biometric templates can decline security of the application.
Attacks on the template can lead to the vulnerabilities like insertion of a fake template, modification of an existing template, removal of an existing template, and replicate the template which can be replayed to the matcher to gain unauthorized access. Authors have made an attempt to develop intelligent tool which assists in detection of vulnerabilities.

The concepts of system misuse and access anomalies have their equivalent counterparts in database intrusions. As a result, there are numerous definitions for the term database intrusion, with variations for different applications, systems, requirements etc. In this proposed model biometric template database intrusion refers to “the act of individuals or groups of individuals who use the database without authorizations, and those who are authorized, but abuse their privileges”.

4.2 Biometric storage

4.2.1. Biometric Template

A template represents a set of salient features that summarizes the biometric data (signal) of an individual. Due to its compact nature, it is commonly assumed that the template cannot be used to elicit complete information about the original biometric signal.

Biometric Templates contain very sensitive information used to identify people which are bound to them. It is the template that is used to determine the user's rights and privileges to access that resource. Each individual’s reference template must be stored in an accessible repository which can be compared to the user’s biometric sample at the time of verification. It is commonly accepted that a biological template cannot be reverse engineered to create a complete biometric input of a user such as creating a complete fingerprint from the biometric input.

A Biometric Template can be stored in a table column as RAW data type, Simple Object data type, XML data type, Full Common Biometric Exchange File Format compliant (CBEFF) data type.

- RAW data type
  These datatypes are intended for binary data or byte strings. RAW data can be indexed. Raw binary data is of length size bytes where maximum size is 2000 bytes. You must specify size for a RAW value. If LONG RAW data
type is used it can store biometric template of variable length up to 2 gigabytes.

- Object types and other user-defined datatypes let you define datatypes that model the structure and behavior of the data in their applications. An object view is a virtual object table. Object types are abstractions of the real-world entities that application programs deal with. An object type is a schema object with three kinds of components:
  - A **name**, which identifies the object type uniquely within that schema.
  - **Attributes**, which are built-in types or other user-defined types. Attributes model the structure of the real-world entity.
  - **Methods**, which are functions or procedures written in PL/SQL and stored in the database, or written in a language like C or Java and stored externally. Methods implement that an operations the application can perform on the real-world entity.

- **XML data type**
  Type can be used like any other user-defined type. XML datatype can be used as the data type of columns in tables and views. Variables of XML data type can be used in PL/SQL stored procedures as parameters, return values, and so on. A number of useful functions that operate on XML content have been provided. Many of these are provided both as SQL functions and as member functions of XMLType. For example, function Extract extracts specific node(s) from an XMLType instance. You can use XMLType in SQL queries in the same way as any other user-defined datatypes in the system. This Oracle-supplied type can be used to store and query XML data in the database. XMLtype has member functions you can use to access, extract, and query the XML data using XPath expressions. XPath is another standard developed by the W3C committee to traverse XML documents. Oracle XMLType functions support many W3C XPath expressions. Oracle also provides a set of SQL functions and PL/SQL packages to create XMLType values from existing relational or object-relational data. Queries and DML on XMLType columns operate the same regardless of the storage mechanism.

- **Full Common Biometric Exchange File Format compliant (CBEFF) data type**
  A Common Biometric Exchange File Format defines a common set of data elements necessary to support these biometric technologies. It promotes
interoperability of biometric-based application programs and systems developed by different vendors by allowing biometric data interchange. These data can be placed in a single file used to exchange biometric information between different system components or between systems. Formatting the data (e.g. allowing individual components to be referenced) will allow an application to easily recognize important processing information about the biometric data such as what type of biometric is available, what version number, vendor’s name, etc. CBEFF accommodates any biometric technology (Podio, et al., 2001). It includes the definition of format and content for data elements such as:

- A biometric data header that contains such information as version number, length of data, whether the data is encrypted or not, etc., for each biometric type available to the application or system;
- Biometric data (content not specified);
- Any other required biometric data or data structures.

A poor quality template or reference can cause considerable problems for the user, often resulting in re-enrollment.

As per application requirement developer of biometric template use any of the above mentioned data types. For the proposed system authors consider Biometric Template stored in the form of RAW data type.

### 4.2.2. Available biometric template storage

Template storage is an area of considerable and growing concern, particularly with large-scale applications that may accommodate hundreds-of-thousands of individuals. The resources to assure the security, quality, maintenance, and management of the data can be formidable and the liability, should the security of the templates be breached, is considerable. Possible template storage options include:

- Central Repository
- A Local Storage with-in the Biometric Reader Device
- On a portable token such as smart card.

Storing the template in a central repository is the option that will most likely occur to IT systems administrators. This may work well in a secure networked environment where there is sufficient operational speed for template retrieval to be invisible to the user. Use of a central data repository also allows more effective use through
network-wide enrollment and disenrollment. While very large central databases raise other concerns discussed elsewhere in this manual, they might be the only efficient way to manage a large identity management system. Care should be taken in system design to ensure that templates are protected when in transit over the network through encryption. Potential disadvantages could be that with a large number of readers working simultaneously, there could be significant data traffic, especially if users are impatient and submit multiple verification/identification attempts. The size of the biometric template itself will have some impact on this issue, with popular methodologies varying between nine bytes and 6Kb. Another aspect to consider is if the network fails, then the system effectively stops, unless there is reliable network backup or some type of additional local/remote storage. This may be possible to implement with some devices using the internal storage on a device or PC for recent cached or localized users and instructing the system to search the central repository if the template cannot be found locally.

Storing the template within the biometric reader device or PC has both advantages and disadvantages, depending on exactly how it is implemented. The advantage is potentially fast operation as a relatively small number of templates may be stored and manipulated efficiently within the device or PC. In addition, there is no reliance on an external process or data link access to the template. In the event of device failure, an alternative device or access point may be substituted as a temporary measure. In some cases where devices may be networked together directly, it is possible to share templates across the network. The potential disadvantage is that templates may be somewhat vulnerable and dependent upon the device being both present and functioning correctly. If anything happens to the device, the template database may need to be re-installed or the user re-enrolled. For templates stored on a hard drive of a personal computer, damage to the disk drive or corrupted data may require re-enrollment of the user.

Storing the template on a token is an attractive option for two reasons. First, it requires no local or central storage of templates and, second, the user carries his/her template with him/her and can use it at any authorized reader device. The template could be stored in the memory of the card or token device or even printed on a card or document in barcode format. Potential disadvantages include the potential loss or damage of the token and the resulting need to re-enroll the user. Additionally, if the user is attracted to the system because he/she believes or was advised that he has
effective control and ownership of his own template, there may be objections to also storing the templates elsewhere in the system. Another potential disadvantage may be unit cost and system complexity if chip card/smart card readers and biometric readers need to be combined at each enrollment and verification station. Finally, if the chip’s operating system and data are successfully hacked, this option could be vulnerable from a security standpoint.

Authors consider biometric template stored within a Central Repository. Central repositories allow users to enroll at a central location and be recognized at any networked biometric device. The privacy risks grow multifold for central storage locations. Central repositories allow for easy auditing of authentication attempts.

### 4.2.3. Vulnerabilities in a Biometric Template Storage

According to common criteria of biometric evaluation methodology supplement (Biometric Evaluation Methodology (BEM) supplement, 2002) it is particularly important to consider that attacks can be done on the direct input and output of a biometric template. One of the most potentially damaging attacks on a biometric system is against the biometric templates stored in the system database.

Attacks on the template can lead to the vulnerabilities like insertion of a fake template, modification of an existing template, removal of an existing template, and replicate the template which can be replayed to the matcher to gain unauthorized access. The Attacks can be done by authorized or unauthorized users. The users abuse of their rights and privileges to do unauthorized activities and to obtain unauthorized access.

![Vulnerabilities in a Biometric Storage Template](image)

**Figure 4.2.3.1: Vulnerabilities in a Biometric Storage Template** (Ratha, Connell, & Bolle, An analysis of minutiae matching strength, 2001)
The authors consider two main categories of users which are normal user and user with DBA role that, intentionally or unintentionally damage the system (Jain, Nandakumar, & Nagar, 2008).

As an example, a fingerprint template copied from a bank’s database may be used to search a criminal fingerprint database or crosslink to person’s health records. The imposter can access centrally stored template to avail unauthorized service whereas a lawful user will face Denial of Service. To avoid this, smart cards are preferred. In that case, the template is stored in write once mode and erased or destroyed if altered. When this scenario is not an option, strong security controls or protection schemes must protect the template.

4.3 Auditing used for Intrusion Detection

4.3.1. Overview of Audit Concept

Auditing is always about accountability, and is frequently done to protect and preserve privacy for the information stored in databases. Concern about privacy policies and practices has been rising steadily with the ubiquitous use of databases in businesses and on the Internet.

The audit record is the only way of alerting a system administrator that a suspicious event has taken place. Auditing is the monitoring and recording of selected user database actions. Auditing is normally used to:

- Investigate suspicious activity. For example, if an unauthorized user is deleting data from tables, the security administrator might decide to audit all connections to the database and all successful and unsuccessful deletions of rows from all tables in the database
- Monitor and gather data about specific database activities. For example, the database administrator can gather statistics about which tables are being updated, how many logical I/Os are performed, or how many concurrent users connect at peak times.

Audit records can contain different types of information, depending on the events audited and the auditing options set.

Although auditing is relatively inexpensive, limit the number of audited events as far as possible. Doing so minimizes the performance impact on the execution of audited statements and the size of the audit trail, making it easier to analyze and understand.
Use the following general guidelines when devising an auditing strategy:

- **Evaluate the purpose for auditing.**
  After you have a clear understanding of the reasons for auditing, you can devise an appropriate auditing strategy and avoid unnecessary auditing. For example, suppose you are auditing to investigate suspicious database activity. This information by itself is not specific enough. What types of suspicious database activity do you suspect or have you noticed? A more focused auditing purpose might be to audit unauthorized deletions from arbitrary tables in the database. This purpose narrows the type of action being audited and the type of object being affected by the suspicious activity.

- **Audit knowledgeably.**
  Audit the minimum number of statements, users, or objects required to get the targeted information. This prevents unnecessary audit information from cluttering the meaningful information and consuming valuable space in the SYSTEM tablespace. Balance your need to gather sufficient security information with your ability to store and process it. For example, if you are auditing to gather information about database activity, and then determine exactly what types of activities you want to track, audit only the activities of interest, and audit only for the amount of time necessary to gather the information that you desire. As another example, do not audit objects if you are only interested in logical I/O information for each session.

The recording of audit information can be enabled or disabled. This functionality allows any authorized database user to set audit options at any time, but reserves control of recording audit information for the security administrator.

### 4.3.2. Auditing: Tool for Intrusion Detection

A fundamental tool for intrusion detection is the audit record. Some record of ongoing activity by users must be maintained as input to an intrusion detection system.

Basically, two audit records are used:

- **Native audit records**
  Virtually all multi-user operating systems include accounting software that collects information on user activity. The advantage of using this information is that no additional collection software is needed. The disadvantage is that
the native audit records may not contain the needed information or may not contain it in a convenient form.

- Detection-specific audit records:
  A collection facility can be implemented that generates audit records containing only that information required by the IDS. One advantage of such an approach is that it could be made vendor independent and ported to a variety of systems. The disadvantage is the extra overhead involved in having, in effect, two accounting packages running on a machine.

A good example of detection-specific audit records is one developed by Dorothy Denning (Dorothy, 1987) each audit record contains the required fields. Most user operations are made up of a number of elementary actions. The decomposition of a user operation into elementary actions has several advantages in simplifying the tracking of activity on the system.

In order to identify normal behavior, administrator uses the database audit files for extracting information regarding user’s actions. The audit records, after being processed, are used to form initial profiles representing acceptable actions. Each entry in the audit files is represented as separate data unit; these units are then combined to form desired profiles.

Auditing tracks the activity of users and processes by recording selected types of events in the logs of a server or workstation. It will provide information required to spot attempted attacks, to investigate what happened when an incident occurred, and to possibly provide evidence in support of an investigation.

The audit records provide input to the intrusion detection function in two ways. First, the designer must decide on a number of quantitative metrics that can be used to measure user behavior. Examples include: counter, interval timer, resource utilization. An analysis of audit records over a period of time can be used to determine the activity profile of the average user. Thus, the audit records serve to define typical behavior. Second, current audit records are the input used to detect intrusion. That is, the intrusion detection model analyzes incoming audit records to determine deviation from average behavior.
4.3.3. Auditing using Oracle

Securing the database against inappropriate activity is only part of total security package. Oracle offers the security administrator on the Oracle database. The other major component of the Oracle security architecture is the ability to monitor database activity to find out suspicious or inappropriate use. Oracle provides this functionality via the use of database auditing.

Auditing is the monitoring and recording of selected user database actions. Auditing is normally used to investigate suspicious activity as well as monitor and gather data about specific database activities.

Audit records can contain different types of information, depending on the events audited and the auditing options set. It includes information such as the user name, the session identifier, the terminal identifier, the name of the schema object accessed, the operation performed or attempted, the completion code of the operation, the date and time stamp, the system privileges used for the operation that was audited.

Audit records can be stored in either a data dictionary table, called the database audit trail, or an operating system audit trail.

Oracle supports three general types of auditing:

- Statement auditing
  
  It is the selective auditing of SQL statements with respect to only the type of statement, not the specific schema objects on which it operates. Statement auditing options are typically broad, auditing the use of several types of related actions for each option. For example, AUDIT TABLE tracks several DDL statements regardless of the table on which they are issued. You can set statement auditing to audit selected users or every user in the database.

- Privilege auditing
  
  It is the selective auditing of, use of powerful system privileges to perform corresponding actions, such as AUDIT CREATE TABLE. Privilege auditing is more focused than statement auditing because it audits only the use of the target privilege. You can set privilege auditing to audit a selected user or every user in the database.
• Schema object auditing

It is the selective auditing of specific statements on a particular schema object, such as AUDIT SELECT ON student. Schema object auditing is much focused, auditing only a specific statement on a specific schema object. Schema object auditing always applies to all users of the database.

To control the Oracle auditing subsystem use system commands such as:

\[
\text{AUDIT SELECT, INSERT, UPDATE, DELETE ON <object_name>}
\]
\[
\text{BY ACCESS WHENEVER SUCCESSFUL;}
\]

Table 4.3.3.1: Audit command

The AUDIT command only turns auditing options on; it does not enable auditing as a whole. To turn auditing on and control whether Oracle generates audit records based on the audit options currently set, set the AUDIT_TRAIL initialization parameter in the database's parameter file.

The database audit trail is a single table named SYS.AUD$ in the SYS schema of each Oracle database's data dictionary. Several predefined views such as DBA_AUDIT_TRAIL and DBA_ROLE_PRIVS are provided to use the information in this table.

It includes information such as the user name, the session identifier, the terminal identifier, the name of the schema object accessed, the operation performed or attempted, the completion code of the operation, the date and time stamp, the system privileges used for the operation that was audited.

The parameter can be set by altering system using following parameters.

\[
\text{Audit_trail = DB}
\]

Enables database auditing and directs all audit records to the database audit trail (SYS.AUD$), except for records that are al-ways written to the operating system audit trail.

\[
\text{Scope=SPFILE}
\]

Uses the SCOPE clause because the database instance had been started using a server parameter file (SPFILE). Starting the database with a server parameter file is the preferred way of starting a database instance.
The operating system audit trail is encoded and not readable, but it is decoded in data dictionary files and error messages. The recording of audit information can be enabled or disabled. This functionality allows any authorized database user to set audit options at any time but reserves control of recording audit information for the security administrator.

Oracle Database provides a depth of auditing that readily enables system administrators to implement enhanced protections, early detection of suspicious activities, and finely-tuned security responses.

4.4 Proposed System

4.4.1 Architecture of proposed system

Authors designed and developed one of the agents of multi-agent system called Biometric Template Storage Intrusion Detection Assistant. Our architecture consists of a user interface module, an inference engine, a knowledgebase of illegal transactions and audit trail of ORACLE database.

Authors consider the simple reflex agent to distinguish the input from their environment i.e. DBA audit trail and interpret it to a state that matches the rules. This approach consists in detecting intrusions exploiting well-known system vulnerabilities. It is based on the fact that any known attack produces a specific trace in the audit trail or in the network data.

Figure: 4.4.1.1: Architecture of Biometric Template storage Intrusion Detection Assistant
This approach works as follows:

- Attacking scenarios are collected,
- These scenarios are translated into facts using some predefined rules.
- Extracted knowledge is utilized to take some decision, an alarm can be raised.
- Using backward chaining approach, source of intrusion can be found out.

Automatic security content updates target specific vulnerabilities and are acquainted with unknown exploits and take preventive action. This intelligent agent is located on the Biometric Template storage database.

For this approach authors have used Knowledge based method (expert systems) which translates attacks that are collected through DBA audit trail of ORACLE. The current data are compared with the predefined rules. If a rule matches, knowledge is created and an alarm is raised. The construction of these rules depends entirely on the expertise of the security officer.

### 4.4.2. Logic used to develop the proposed system

Authors consider two main categories of users as, normal user and user with DBA role that, intentionally or unintentionally damage the system.

In this approach authors consider that datatype of biometric template stored in a table is RAW datatype with size 1024 bytes. Authors have consider the database table student where the biometric data is stored having following structure. To create student table the following command is used.

```sql
CREATE TABLE Student
  (Roll_no VARCHAR2(10) ,
   name VARCHAR2(50),
   course VARCHAR2(20),
   fingerprint_template RAW(1024));
```

**Table 4.4.2.1: Create table to store biometric template data**

If any type of user can try insertion, deletion, modification or copy contents to or from the table then data about these transaction is recorded in different views of oracle.

Authors collect suspicious data from DBA_AUDIT_TRAIL by firing SQL query on DBA_AUDIT_TRAIL and DBA_ROLE_PRIVS views.
The SQL query which is used to collect audited data about normal user is as follows:

```sql
SELECT username, userhost, action_name, owner, obj_name, os_username,
       extended_timestamp, sessionid
FROM dba_audit_trail
WHERE obj_name='STUDENT'
minus
SELECT username, userhost, action_name, owner, obj_name,
       os_username, extended_timestamp, sessionid
FROM dba_audit_trail a, (SELECT * from dba_role_privils b
       WHERE granted_role='DBA')b
WHERE a.username = b.grantee;
```

Table 4.4.2.2: SQL query which is used to collect audited data about normal user

The SQL query which is used to collect audited data about DBA user is as follows:

```sql
SELECT dat.username, dat.userhost, dat.action_name,
       dat.extended_timestamp, dat.owner, dat.obj_name,
       dat.OS_username, dat.sessionid, drp.granted_role
FROM  dba_audit_trail dat, dba_role_privils drp
WHERE  obj_name = 'STUDENT' and dat.username=drp.grantee and
       drp.granted_role="DBA";
```

Table 4.4.2.3: SQL query which is used to collect audited data about DBA user

The database is accessed using the JDBC (Java Database Connectivity). The result set is asserted into facts.

Authors collect suspicious data from DBA_AUDIT_TRAIL, DBA_ROLE_PRIVS and create different JESS templates as per requirement which contains action message, action name, username, userhost, timestamp etc.

The details about illegal transaction whose actions were audited like name of the user (USERNAME), operating system login username of the user (OS_USERNAME), client host machine name (USERHOST), Numeric ID of each ORACLE session (SESSIONID), Name of the object affected by action (OBJ_NAME), creator of the object affected by the action (OWNER), Timestamp of the creation of the audit trail entry in Coordinated Universal Time (UTC) zone (EXTENDED_TIMESTAMP) is stored into JESS in terms of facts.
In our implementation, one of the samples of JESS template which stores the suspicious data has following definitions.

```
(deftemplate Trans
(slot actmessage)
(slot action_name)
(slot username)
(slot userhost)
(slot timestamp)
......................)
```

Table 4.4.2.4: Jess Template to store audit trail

In addition to the facts, rules are defined. Authors design different rules for suspicious transactions like insert, modify, remove and copy the biometric template storage. A suspicious knowledge is stored as a form of facts and rules in a JESS knowledge base. It is somewhat similar to a relational database, especially in that the facts must have a specific structure.

Authors design the rules in such a way that when rules are fired some new facts are asserted for counting of the suspicious transactions, suspicious users, suspicious hosts and some facts are modified. Java and Jess are used for development.

4.4.3. Back tracing used for source detection

In a backwards chaining system, rules are still if..then statements, but the engine seeks steps to activate rules whose preconditions are not met. This behavior is often called "goal seeking". JESS supports both forward and backward chaining. Authors use back tracing for postmortem of the intrusion to find source of intrusion. They use Defquery construct for back tracing, which displays detail knowledge about OS username, username, object name, owner of object, time stamp, session-id and so on.

The Defquery construct lets you create a special kind of rule with no right-hand-side. While rules act spontaneously, queries are used to search the knowledge base under direct program control. A rule is activated once for each matching set of facts, while a query gives you a java.util.Iterator of all the matches. It can be convenient to use queries as triggers for backward chaining. For this to be useful, Rete.run() must be called while the query is being evaluated, to allow the backward chaining to occur.
Facts generated by rules fired during this run may appear as part of the query results.

Authors use Defquery as follows:

```
```

**Table 4.4.3.1: defquery used for back tracing**

The details are collected about illegal transaction whose actions were audited like name of the user (USERNAME), operating system login username of the user (OS_USERNAME), client host machine name (USERHOST), Numeric ID of each ORACLE session (SESSIONID), Name of the object affected by action (OBJ_NAME), creator of the object affected by the action (OWNER), Timestamp of the creation of the audit trail entry in Coordinated Universal Time (UTC) zone (EXTENDED_TIMESTAMP) using backward chaining approach.

**4.4.4. Encoding the Rules used for this agent**

As per Literature review authors have defined rules for vulnerabilities in biometric template storage like insertion of a fake template, modification of an existing template, removal of an existing template, and replicate the template which can be replayed to the matcher to gain unauthorized access.

A JESS rule is something like if…then statement in a procedural language, but it is not used in a procedural way. While if…then statements are executed at a specific time and in a specific order, according to how the programmer writes those, JESS rules are executed whenever their if parts (their left-hand-sides or LHSs) are satisfied, given only that the rule engine is running. This makes JESS rules less deterministic than a typical procedural program. Rules are defined in JESS using the Defrule construct. Defrule means Defines a new rule. Defrule can search knowledge base to find relationships between facts, and rules can take actions based on the contents of one or more facts.

Jess rules only fire while the rule engine is running (although they can be activated while the engine is not running.) To start the engine running, we issue the `run()` command.
A sample of the design rule is as follows:

**If** action name is insert **Then** modify action message as Illegal Insertion and increase count of insert actions and assert counted value into facts and assert username who did insert action into facts and assert hostname from which Insert action take place into facts.

The following is the JESS language representation of the above rule.

```jess
(defrule insert_rule
  ?r1<- (Trans (action_name ?*actname*) (username ?un) (userhost ?uh))
  ?r2<- (Cnt_action (action_name ?*actname*))
  ?c1<- (accumulate (bind ?*cnt* 0) (bind ?*cnt* (+ ?*cnt* 1)) ?*cnt* (Trans (action_name ?a&: (eq ?a \INSERT))))
  => (modify ?r1 (actmessage \Illegal-Insertion\))
  (modify ?r2 (cnt ?*cnt*))
  (assert (Cnt_user (username ?un)))
  (assert (Cnt_host (userhost ?uh)))
  (assert (Priority (usertype \NormalUser\) (username ?un) (userhost ?uh) (action_name ?*actname*))))
```

**Table 4.4.4.1: Jess rule to check Insert action of Normal user**

The above rule states that **If** normal user inserted data into table **Then** modify action message as Illegal Insertion and increase count of insert actions and assert counted value into facts and assert username who did insert action into facts and assert hostname from which Insert action take place into facts. Assert usertype into Priority fact.

```jess
(defrule delete_rule
  ?r3<- (Trans (action_name ?*actname1*) (username ?un) (userhost ?uh))
  ?r4<- (Cnt_action (action_name ?*actname1*))
  ?c2<- (accumulate (bind ?*cnt* 0) (bind ?*cnt* (+ ?*cnt* 1)) ?*cnt* (Trans (action_name ?a&: (eq ?a \DELETE))))
  => (modify ?r3 (actmessage \Illegal-Deletion\))
  (modify ?r4 (cnt ?*cnt*))
  (assert (Cnt_user (username ?un)))
  (assert (Cnt_host (userhost ?uh)))
  (assert (Priority (usertype \NormalUser\) (username ?un) (userhost ?uh) (action_name ?*actname1*)))))
```

**Table 4.4.4.2: Jess rule to check Delete action of normal user**

The above rule states that **If** normal user deleted data from table **Then** modify action message as Illegal Deletion and increase count of delete actions and assert counted value into facts and assert username who did delete action into facts and assert
hostname from which delete action take place into facts. Assert usertype into Priority fact.

Table 4.4.4.3: Jess rule to check Update action of normal user
The above rule states that If normal user updated data from table Then modify action message as Illegal Updation and increase count of update actions and assert counted value into facts and assert username who did update action into facts and assert hostname from which update action take place into facts. Assert usertype into Priority fact.

Table 4.4.4.4: Jess rule to check copy action of normal user
The above rule states that If normal user copied data from table Then modify action message as Illegal copy and increase count of copy actions and assert counted value into facts and assert username who did copy action into facts and assert hostname from which copy action take place into facts. Assert usertype into Priority fact.
Table 4.4.4.5: Jess rule to check Insert action of DBA

The above rule states that If DBA inserted data into table Then modify action message as Illegal Insertion and increase count of insert actions and assert counted value into facts and assert username who did insert action into facts and assert hostname from which Insert action take place into facts. Assert usertype into Priority fact.

(defrule dbainsert_rule ?rd1<- (dbaTrans (action_name ?*dactname*)) (username ?un) (userhost ?uh))
?rd2<- (Cnt_dbaaction (action_name ?*dactname*))
?cd1<- (accumulate (bind ?*dbacnt* 0) (bind ?*dbacnt* (+ ?*dbacnt* 1))
?*cnt* (dbaTrans (action_name ?a&: (eq ?a INSERT))))
=> (modify ?rd1 (actmessage Illegal-Insertion))
(modify ?rd2 (cnt ?*dbacnt*)) (assert (Cnt_dbauser (username ?un)))
(assert (Cnt_dbahost (userhost ?uh)))
(assert (Priority (usertype \DBA)+
(username ?un) (userhost ?uh) (action_name ?*dactname*))))

Table 4.4.4.6: Jess rule to check Delete action by DBA

The above rule states that If DBA user deleted data from table Then modify action message as Illegal Deletion and increase count of delete actions and assert counted value into facts and assert username who did delete action into facts and assert hostname from which delete action take place into facts. Assert usertype into Priority fact.

(defrule dbadelete_rule ?rd3<- (dbaTrans (action_name ?*dactname1*)) (username ?un) (userhost ?uh))
?rd4<- (Cnt_dbaaction (action_name ?*dactname1*))
?cd2<- (accumulate (bind ?*dbacnt* 0) (bind ?*dbacnt* (+ ?*dbacnt* 1))
?*cnt* (dbaTrans (action_name ?a1&: (eq ?a1 DELETE))))
=> (modify ?rd3 (actmessage Illegal-Deletion))
(modify ?rd4 (cnt ?*dbacnt*)) (assert Cnt_dbauser (username ?un)))
(assert (Cnt_dbahost (userhost ?uh)))
(assert (Priority (usertype \DBA)+
(username ?un) (userhost ?uh) (action_name ?*dactname1*))))
Table 4.4.4.7: Jess rule to check Update action by DBA

The above rule states that If DBA updated data from table Then modify action message as Illegal Updation and increase count of update actions and assert counted value into facts and assert username who did update action into facts and assert hostname from which update action take place into facts. Assert usertype into Priority fact.

(defrule dbaupdate_rule  ?rd5<- (dbaTrans (action_name ?*dactname2*))
  (username ?un) (userhost ?uh))
?rd6<- (Cnt_dbaaction (action_name ?*dactname2*))
?cd3<- (accumulate (bind ?*dbacnt* 0) (bind ?*dbacnt* (+ ?*dbacnt* 1)))
?dbacnt* (dbaTrans (action_name ?a1 &= (eq ?a1 UPDATE)))
=> (modify ?rd5 (actmessage \Illegal-Updation\))
(modify ?rd6 (cnt ?*dbacnt*)) (assert (Cnt_dbauser (username ?un)))
(assert(Cnt_dbahost (userhost ?uh )))
(assert (Priority (usertype \DBA\))
(username ?un) (userhost ?uh) (action_name ?*dactname2*))

Table 4.4.4.8: Jess rule to check copy action by DBA

The above rule states that If DBA copied data from table Then modify action message as Illegal copy and increase count of copy actions and assert counted value into facts and assert username who did copy action into facts and assert hostname from which copy action take place into facts. Assert usertype into Priority fact.

(defrule dbaselect_rule  ?rd7<- (dbaTrans (action_name ?*dactname3*))
  (username ?un) (userhost ?uh))
?rd8<- (Cnt_dbaaction (action_name ?*dactname3*))
?cd4<- (accumulate (bind ?*dbacnt* 0) (bind ?*dbacnt* (+ ?*dbacnt* 1)))
?*dbacnt* (dbaTrans (action_name ?a1 &= (eq ?a1 SELECT)))
=> (modify ?rd7 (actmessage \Illegal-Copy\))
(modify ?rd8 (cnt ?*dbacnt*)) (assert (Cnt_dbauser (username ?un)))
(assert(Cnt_dbahost (userhost ?uh )))
(assert (Priority (usertype \DBA\))
(username ?un) (userhost ?uh) (action_name ?*dactname3*))
Table 4.4.4.9: Jess rule to count illegal actions by normal user

The above rule states that number of normal users who try to attempt illegal action is counted.

(defrule Countuserrule ?r11<- (Trans (username ?un))
  ?r12<- (Cnt_user (username ?un)) ?cr1<- (Priority (username ?un) (usertype ?"ut"))
  c11<- (accumulate (bind ?"ucnt" 0) (bind ?"ucnt" (+ ?"ucnt" 1))
    ?"ucnt" (Trans (username ?u1&:(eq ?u1 ?un))))
  => (modify ?r12 (cnt ?"ucnt"))
  (modify ?cr1 (userfrq ?"ucnt")))

Table 4.4.4.10: Jess rule to count hosts used for illegal actions by normal user

The above rule states that number of hosts from which normal user try to attempt illegal action is counted.

(defrule Counthostrule ?r21<- (Trans (userhost ?uh))
  r22<- (Cnt_host (userhost ?uh)) ?cr2<- (Priority (userhost ?uh) (usertype ?"ut"))
  c21<- (accumulate (bind ?"hcnt" 0) (bind ?"hcnt" (+ ?"hcnt" 1))
    ?"hcnt" (Trans (userhost ?h1&:(eq ?h1 ?uh))))
  (modify ?r22 (cnt ?"hcnt"))
  modify ?cr2 (hostfrq ?"hcnt")))

Table 4.4.4.11: Jess rule to count illegal actions by DBA

The above rule states that number of DBA who try to attempt illegal action is counted.

(defrule Countdbauserrule ?rd11<- (dbaTrans (username ?un))
  ?rd12<- (Cnt_dbauser (username ?un)) ?crd2<- (Priority (username ?un))
  c11<- (accumulate (bind ?"dudcnt" 0) (bind ?"dudcnt" (+ ?"dudcnt" 1))
    ?"dudcnt" (dbaTrans (username ?u1&:(eq ?u1 ?un))))
  => (modify ?rd12 (cnt ?"dudcnt"))
  (modify ?crd2 (userfrq ?"dudcnt")))
Table 4.4.4.12: Jess rule to count hosts used for illegal actions by normal user


The above rule states that number of DBA who try to attempt illegal action is counted.

Similarly there are defined rules for suspicious transactions like modify, remove and copy the biometric template for both normal user and DBA role user.

4.5 Findings of Agent 1

The Biometric Template Storage Intrusion Detection Assistant is nothing but dashboard which displays two tables namely User Intrusion which contains suspicious activities of normal users and DBA Intrusion which contains suspicious activities of DBA.

The result screens as output of our system are shown in figure 4.5.1 and fig 4.5.2. The Biometric Template Storage Intrusion Detection Assistant which displays two tables namely User Intrusion which contains suspicious activities of normal users and DBA intrusion which contains suspicious activities of DBA. A text pane is used to display detail information of selected suspicious activity. Three tables which show top intruders, top suspicious hosts and top suspicious DBA hosts. These tables are used to find out most suspicious user or host and that knowledge is used for taking any preventive actions. One bar graph shows which transaction is done repeatedly as suspicious activity by normal user while another one that of DBA. User intrusion graph shows Insert transaction tried at 9 times, Delete transaction tried at 19 times, Update trans-action tried at 18 times, Delete transaction tried at 20 times as suspicious activity. DBA intrusion graph shows Insert transaction tried at 6 times, Delete transaction tried at 7 times, Update trans-action tried at 2 times, Delete transaction tried at 8 times as suspicious activity. If user selects any row from normal user suspicious activity table, then details about name of the user whose action where audited (USERNAME), operating system login username of the user whose actions were audited(OS_USERNAME), client host machine name (USERHOST),
Numeric ID of each ORACLE session (SESSIONID), Name of the object affected by action (OBJ_NAME), Timestamp of the creation of the audit trail entry in Universal Time Coordinated (UTC) zone (EXTENDED_TIMESTAMP) will display on the screen. Fig. 4.5.1 shows detail information about selected row of user Intrusion table which contains suspicious activity of normal user. Similarly if user selects any row from DBA suspicious activity table, then details about name of the DBA whose action were audited (USERNAME), operating system login username of the user whose actions were audited (OS_USERNAME), client host machine name (USERHOST), Numeric ID of each ORACLE session (SESSIONID), Name of the object affected by action (OBJ_NAME), Timestamp of the creation of the audit trail entry in Universal Time Coordinated (UTC) zone (EXTENDED_TIMESTAMP) will display on the screen. Figure 4.5.2 shows detail information about selected row of DBA intrusion table which contains suspicious activity of DBA.

If user selects any row from normal user suspicious activity table or DBA suspicious activity table, then details about name of the user whose action were audited, operating system login username of the user whose actions were audited, client host machine name, Numeric ID of each ORACLE session, Name of the object affected by action, Timestamp of the creation of the audit trail entry in Universal Time Coordinated (UTC) zone will display on the screen.

4.6 Prevention Technique suggested

1. Security administrator and database administrator sets proper database security techniques and triggers for transactions to block suspicious user or suspicious host can be used.

2. Use techniques like encryption to avoid the misuse of stored biometric template. (Encryption technique is described in chapter 8)

A security administrator finds priorities of detected intrusion. It is very easy to him/her to prevent those suspicious actions, suspicious users and suspicious hosts. (Priorities for prevention is depicted in chapter 5)
Figure: 4.5.1: Dashboard design and source detection for Normal user
Figure: 4.5.2: Dashboard design and source detection for DBA user
CONCLUDING REMARKS

In this chapter, a simple implementation of knowledge based Biometrics Template storage Intrusion Detection assistant is portrayed. This intelligent agent is located on the Biometric Template storage database. This proposed intelligent agent considered biometric template database is stored in the central repository. The intrusion detection is executed in back-ground. When it detects suspicious or illegal activities, it notifies the security administrator. For detecting intrusive activities, IDS can use audit trail data. It performs intrusion detection using Operating System’s audit trail, RDBMS audit trail or information from multiple monitored hosts. The system consists of a user interface module, an inference engine, a knowledgebase of illegal transactions and audit trail of ORACLE database. Inference engine is implemented using JESS which is a Java Expert System Shell. User interface is developed using Java.