A Novel Clustering Algorithm for Database Anomaly Detection

Jinkun Geng[1(✉)], Daren Ye[1], Ping Luo[2], and Pin Lv[3]

[1] School of Software, Beihang University, Beijing 100191, China
steam1994@163.com
[2] Key Laboratory for Information System Security, Ministry of Education, Tsinghua National Laboratory for Information Science and Technology(TNlist), School of Software, Tsinghua University, Beijing 100084, China
luop@mail.tsinghua.edu.cn
[3] State Information Center, Beijing 100045, China

Abstract. As a main method in database intrusion detection, database anomaly detection should be able to detect users' operational behaviours for timely prevention of possible attacks and for guarantee of database security. Aiming at this, we apply cluster analysis techniques to anomaly detection and propose a novel density-based clustering algorithm called DBCAPSIC, which is adopted to clustering database users according to their behavior types and behavior frequencies. Privilege patterns are extracted from the clusters and serve as a reference in anomaly detection. The simulation experiment proves that the algorithm can recognize the anomalous operations with few mistakes.

Keywords: Database anomaly detection · Database security · Cluster analysis · Privilege pattern

1 Introduction

Computer science and network technology are developing rapidly, leading to data explosions in almost every field. Data has become an important asset today and database security is gaining more and more attention. [8-9] [14] As a crucial part in database security protection, database intrusion detection should be able to detect users' operational behaviours for timely prevention of possible attacks.

However, there is currently few intrusion detection researches focusing on database and the built-in security mechanisms are far from effective to detect and prevent anomalous behaviour of applications and intrusions from attackers [8-9].The existing intrusion detection systems are insufficient to make ideal intrusion detection for databases. Therefore, the study of intrusion detection aiming at databases, especially the anomaly detection, is of great significance both theoretically and practically.

Data mining techniques are widely adopted in the fields of business, medicine, education and engineering [10,19-23] because of the capability of discovering lots of useful knowledge automatically in the analysis of massive information. This inspire us to adopt some of the methods in our research.

B. Thuraisingham et al. (Eds.): SecureComm 2015, LNICST 164, pp. 682–696, 2015.
DOI: 10.1007/978-3-319-28865-9_45

In this paper, we focus on database anomaly detection and propose a "density-based clustering algorithm via pre-sampling and inferior centroid" (denoted as DBCAPSIC). We embed DBCAPSIC into the anomaly detection algorithm. With the privilege patterns extracted from clusters generated by DBCAPSIC, we can detect the real-time operations on the monitored DBMS and recognize the anomalous operations.

The rest of the paper is organized as follows: Section 2 introduces some preliminary concepts in the field of database intrusion detection. Section 3 first illustrates proposes the algorithm DBCAPSIC and states the anomaly detection method based on DBCAPSIC. Section 4 presents the experiment and analyses the experimental result. Section 5 concludes the whole paper.

2 Preliminaries

Before the description of the algorithm, we first provide some preliminary definitions in this section.

2.1 Definitions of Objects

Definition 1 The 2-tuple consisting of a database object and an operational behaviour type is defined as a behaviour pattern (*BP*),i.e.

$$BP = \{object, type\},$$

where *object* is a database object such as a table, a view and so on, and *type* is the type of the behaviour operated on *object*, such as *SELECT, UPDATE* and so on.

Definition 2 The 3-tuple consisting of a database object, an operational behaviour type and a frequency value of the behaviour is defined as a behaviour object (*BO*), i.e.

$$BO = \{object, type, frequency\},$$

where *frequency* is the times of the behaviour of *type* operated on *object* by a certain database user in a period of history. For example, if a user has made 3 *SELECT* operations on the table *discount*, then we get a *BO* like this:

$$\{discount, SELECT, 3\}.$$

Definition 3 The 2-tuple consisting of a database user and its corresponding behaviour object set (*BOS*) is defined as a user object (*UO*), i.e.

$$UO = \{user, BOS\},$$

where *user* the database user, usually represented by the user's account name, and *BOS* is the set of behaviour objects affiliated to the same user, i.e.

$$BOS = \{BO_1, BO_2, ..., BO_n\}.$$

For the convenience of the following narration, the definition of frequency function is given here.

Definition 4 The frequency function *F(user,BP)* is defined as the operational times (frequency) of operated by *user* in a period of history.

Obviously for a certain behaviour object which is affiliated to *user*, let's say $BO_0 = \{object_0, type_0, frequency_0\}$, the following condition is satisfied that

$$frequency_0 = F(user, BP_0),$$

where $BP_0 = \{object_0, type_0\}$.

With the definition of frequency function we can have

$$BO_0 \Leftrightarrow \{BP_0, F(user, BP_0)\},$$

which is called the behaviour object's 2-tuple definition.

Furthermore, for the behaviour object set *BOS* affiliated to *user*, we can get the behaviour object set's 2-tuple definition:

$$BOS \Leftrightarrow \{BPS, FS\},$$

where $BPS = \{BP_1, BP_2, ..., BP_n\}$ and $FS = \{F(user, BP_1), F(user, BP_2), ...F(user, BP_n)\}$.

2.2 Definition of Measurements

Definition 5 Let UO_1 and UO_2 denote two user objects:

$$UO_1 = \{user_1, BOS_1\}, UO_2 = \{user_2, BOS_2\},$$

where BOS_1 and BOS_2 are represented with the 2-tuple definition, i.e.

$$BOS_1 = \{BPS_1, FS_1\}, BOS_2 = \{BPS_2, FS_2\}.$$

Then the similarity function is defined as

$$similarity(UO_1, UO_2) = \frac{\sum_{a \in BPS_1 \cap BPS_2} \min(\frac{F(user_1, a)}{F(user_2, a)}, \frac{F(user_2, a)}{F(user_1, a)})}{\max(|BPS_1|, |BPS_2|)},$$

where $|BPS|$ refers to the capacity of *BPS*.

Definition 6 The distance function is defined as

$$dist(UO_1, UO_2) = 1 - similarity(UO_1, UO_2),$$

From the definition, we know $0 \le dist \le 1$, and the more similar the two use objects are, the smaller the distance value is, and vice versa. Specially, when the distance value reaches 1, the two user objects have completely different *BPSs*, and they represent database users of different classes.

3 Anomaly Detection with DBCAPSIC

3.1 Basic Idea of DBCAPSIC

The algorithm DBCAPSIC learns from the idea of k-means-type algorithms and density-based clustering algorithms, and it avoids the user-defined cluster numbers and the random selection of starting points, thus overcoming classic k-means algorithm's susceptibility to initial conditions; Moreover, via "pre-sampling method", DBCAPSIC managed to reduce the time complexity to $O(n)$,and by introducing the concept of "inferior-centroid", it solves the "Clustering Failure" problem which common density-based clustering algorithms will meet on certain cases.

For the convenience of the following narration, here we provide definitions of density and radius.

Definition 7 For the data set denoted as $E = \{x_1, x_2, \ldots, x_n\}$, the elements' radius is defined as

$$radius = \frac{2}{n(n-1)} \sum_{i=1}^{n} \sum_{j=i+1}^{n} dist(x_i, x_j),$$

where n is the capacity of E, that is $n = |E|$.

Definition 8 The element's density is defined as

$$\forall x \in E, density(x) = \sum_{j=1}^{n} sign(radius - dist(x, x_j)),\text{where } sign(x) = \begin{cases} 1, \text{x} \geq 0 \\ 0, \text{x} < 0 \end{cases}.$$

3.2 Inferior-Centroid to Avoid "Clustering Failure"

The initial algorithm we adopted runs like this: firstly we calculate the distance of each pair of objects and then we get radius; with radius we can calculate the density of each object to construct density set, i.e.

$$densitySet = \{density(UO) \mid UO \in UOS\}.$$

We choose UO_m from *UOS*, which has the largest density in *densitySet*,as a centroid. With UO_m, we can find all objects within the radius of UO_m and remove their densities from *densitySet*. We will repeat the process until *densitySet* becomes an empty set. Then we get the *centroidSet*. Next,we assign each object to its nearest centroid and get *clusterSet*. Finally we reset the centroid of each cluster to select the object with the largest density to represent the cluster.

During the process of clustering with the density-based algorithms, we discover the following problems: The first problem is the high time complexity which reaches $O(n^2)$thus meeting a bottleneck when dealing with massive data. The second problem is the "Clustering Failure" —Some of the final clusters may contain user objects

that are complete different, that is to say, different classes of *UO*s are assigned into the same cluster.

As for the first problem, we can use "pre-sampling method" to reduce the high time complexity. But for the second problem of "Clustering Failure" , we think it is the defect of common density-based algorithm.

The cause of "Clustering Failure" is because when we judge whether an object *UO* is "qualified" to be added into the cluster $cluster_i$, we just consider the similarity between *UO* and the current centroid $centroid_i$ and ignore the similarity between *UO* and other objects that already have the "quality" to join $cluster_i$. Therefore, it may cause that objects that belong to the same cluster have low similarity or no similarity at all.

To avoid "Clustering Failure", we propose the definition of "inferior-centroid".

Definition 9 Inferior-centroid is the element in the cluster that is farthest from the centroid of the cluster, i.e. the object x is the inferior-centroid of *cluster iff* $x \in cluster$, and $dist(centeroid, x) = \max\left(\left\{dist(centeroid, x_i) \mid x_i \in cluster\right\}\right)$.

3.3 Description of DBCAPSIC Algorithm

In DBCAPSIC, "pre-sampling method" is adopted to reduce the time complexity to a linear level and inferior-centroid is introduced to avoid "Clustering Failure".

The description of DBCAPSIC is shown in Algorithm 1.

Algorithm 1. DBCAPSIC Based on behaviour Type and behaviour Frequency

1.Input:

UOS---The database user object set ,i.e. $UOS = \{UO_1, UO_2, \cdots, UO_n\}$.

λ ---The artificially specified merge coefficient, which belongs to the interval [0,1].

2.Radius Calculation.

2.1 Pre-Sampling. Make a simple random sampling of *UOS* and get the sample set *sUOS*, with the capacity of $\lfloor \sqrt{n} \rfloor$ ([ ] means to round down).

2.2 Radius Estimation. The radius of *sUOS* ,denoted as *sradius* can be calculated and it serves as the approximated radius for *UOS*, i.e. $radius = sradius$.

3.Density Calculation.

$\forall UO \in sUOS$,calculate *density(UO),* and $densitySet = \{density(UO) \mid UO \in sUOS\}$.

4.Center User Object Selection.

Initialize the center user object set *cUOS* and the inferior center user set *iUOS*, i.e. $cUOS = \varnothing$, $iUOS = \varnothing$.

While $densitySet \neq \varnothing$,execute the following sub-steps:

4.1 Select the *UO* with the largest density, denoted as UO_m, then $cUOS = cUOS \cup \{UO_m\}$, $densitySet = densitySet - \{d_m\}$;the current center user object $cUO=UO_m$,the current inferior center user object *iUO* is null.

4.2 $\forall UO_i \in UOS$,

if $UO_i \in cUOS$ or $UO_i \in iUOS$, then traverse the next user object UO_{i+1} ;

else if *dist(cUO,UO$_i$)>radius*, turn to Step 4.3;

else we have $dist(cUO, UO_1) \le radius$, then execute the following sub-sub-steps:

4.2.1 If *iUO* is null, make UO_i the current inferior-centroid and put it into $iUOS$, i.e. $iUO = UO_i \; iUOS = iUOS \cup \{UO_i\}$,

and remove its density from *densitySet* , i.e.

$densitySet = densitySet - \{density(UO_i)\}$, then turn to Step 4.3.

4.2.2 If *iUO* is not null, then judge whether it is satisfied that $dist(iUO, UO_i) \le radius$ .If so, turn to Step 4.2.3, else turn to Step 4.3.

4.2.3 If *dist(cUO,UO$_i$)>dist(cUO,iUO)*,then update the current inferior-centroid, i.e. $iUOS = iUOS - \{iUO\}$, $iUO = UO_i \; iUOSet = iUOSet \cup \{iUO\}$,

then turn to Step 4.3;else directly turn to Step 4.3.

4.3 Repeat Step 4.1 and Step 4.2 until $densitySet = \varnothing$, then we get the center user object set $cUOS$ and the inferior center user set $iUOS$.

5.User Objects Clustering.

5.1 Combine the center user object set with the inferior center user object set, i.e. $cUOS = cUOS \cup iUOS$.

5.2 Initialize a cluster for each center user object in $cUOS$, then we get the cluster set, that is $clusterSet = \{cluster_1, cluster_2, \ldots, cluster_r\}$, where $r = |cUOSet|$ and $\forall cluster_i \in clusterSet, cluster_i = \varnothing$.

5.3 $\forall UO \in UOS$,traverse $cUOS$ and find a certain center user object cUO_j that satisfies $dist(UO, cUO_j) = \min(\{dist(UO, cUO) \mid cUO \in cUOS\})$, cUO_j is the nearest center user to UO ,then remove the UO from UOS and assign it into the cUO_j 's cluster, i.e. $UOS = UOS - \{UO\}$, $cluster_j = cluster_j \cup \{UO\}$.

5.4 After Step 5.3 has finished, we get $UOS = \varnothing$, which means each user object has been assigned to a cluster, then we get the *clusterSet* as the initial clustering result.

6.Cluster Merging.

6.1 Set the merge threshold, i.e. $MergeValue = \lambda \times radius$.

6.2 Select the two object cUO_1 and cUO_2 from $cUOS$ that have the largest similarity. If $dist(cUO_1, cUO_2) \ge MergeValue$, which means the two clusters are still not so similar to merge, then Step 6 has finished, turn to Step 7; else merge the two clusters and remove the two center user objects from $cUOS$, i.e. $cluster_3 = cluster_1 \cup cluster_2$, $clusterSet = clusterSet - \{cluster_1, cluster_2\}$,

$cUOS = cUOS - \{cUO_1, cUO_2\}$ .Then execute Step 6.3.

6.3 Select a new centroid for $cluster_3$. Make a random sampling of $cluster_3$ and get the sample set $scluster_3$ with the capacity of $\left\lfloor \sqrt{|cluster_3|} \right\rfloor$, calculate the average distance between objects in $scluster_3$ and make it the radius of $scluster_3$, named clt_radius ,with it we can calculate the objects' density in $scluster_3$, choose the user object with largest density in $scluster_3$ as the approximate optimal center user object, named cUO_3 ,then cUO_3 is selected as the center user object of $cluster_3$, i.e. $clusterSet = clusterSet \cup \{cluster_3\}$, $cUOS = cUOS \cup \{cUO_3\}$, then return to Step 6.2.

7.Center User Object Adjusting.

7.1 Re-initialize $cUOS$, i.e. $cUOS = \varnothing$.

7.2 Pre-Sampling.

$\forall cluster_i \in clusterSet$, make a random sample of $cluster_i$, the sample set is $scluster_i$ with the capability of $\left\lfloor \sqrt{|cluster_i|} \right\rfloor$.

7.3 Cluster Radius Calculation.

For each $scluster_i$, calculate the approximated radius for $cluster_i$, i.e.

$$clt\_r_i = \frac{2}{n_i(n_i-1)} \sum_{j=1}^{n_i} \sum_{k=j+1}^{n_i} dist(UO_j, UO_k),$$

where $UO_j \in scluster_i, UO_k \in scluster_i$ and n_i is the capability of $scluster_i$, i.e. $n_i = |scluster_i| = \left\lfloor \sqrt{|cluster_i|} \right\rfloor$.

7.4 User Object Density Calculation.

$$\forall UO \in cluster_i, clt\_density(UO) = \sum_{j=1}^{n_i} sign\left(clt\_r_i - dist(UO, UO_j)\right).$$

7.5 $\forall cluster_i \in clusterSet$,choose the user object from its sample set $scluster_i$ with the maximum density , the user object is set as the final center user object of $cluster_i$, i.e. $cUO_i = UO_0$, $cUOS = cUOS \cup \{UO_0\}$.

8.Output;

The user object cluster set $clusterSet$ and the center user object set $cUOS$.

3.4 Analysis of DBCAPSIC

We adopt "pre-sampling method" in Step 2, Step 6.3 and Step 7.5 to get a linear time complexity (the time complexity is $O\left(\left\lfloor \sqrt{n} \right\rfloor^2\right) = O(n)$).

We will explain how the inferior-centroid is adopted to solve the problem of Clustering Failure" .

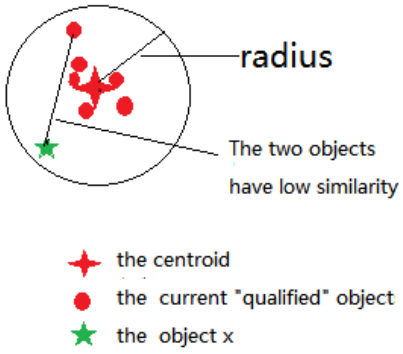

Fig. 1. Clustering without inferior-centroid

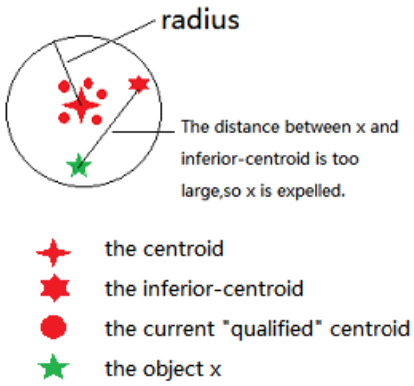

Fig. 2. Clustering with inferior-centroid

In Fig.1, without the inferior-centroid, when we judge whether the object x has the quality to join the current cluster, we just refer to the distance between x and centroid but ignore the distance between x and other "qualified" objects, thus may causing that the objects belonging the same cluster to have low similarity or no similarity at all.

In Fig.2, we introduce inferior-centroid to assist in judging whether x has the quality to join the current cluster. From Fig. 2 we can discover that although x still falls within the radius of the current centroid, but because the distance between x and the current inferior-centroid is so large that x is expelled from the cluster. Thus the problem of "Clustering Failure" is avoided.

After we have introduced the inferior-centroid to assist in judging, we may get clusters like Fig.3 and Fig.4.

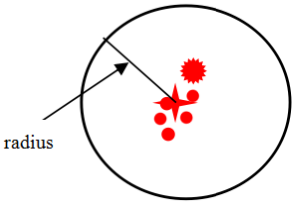

Fig. 3. Cluster with one actual classes

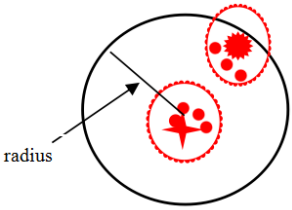

Fig. 4. Cluster with two actual classes

In the process of clustering, the distance between the centroid and the inferior-centroid remains a small value (even 0 in extreme situations), then the centroid and the inferior-centroid are objects of great similarity, besides, the inferior-centroid is the farthest object away from the centroid in the cluster, therefore, it can be inferred that all objects in the cluster are very similar to each other. In this case, the cluster contains just one actual class. We denote it as "One-Class-Clusters" (Fig.3).

In the process of clustering, the inferior-centroid gradually moves away from the centroid with the update. Finally the distance may be close to *radius* . The objects in the cluster may be divided into two classes, some of them are closer to the centroid while the others are closer to the inferior-centroid. We denote it as "Two-Class-Cluster" (Fig.4).

In summary, with inferior-centroids, the clusters can have at most two actual classes of elements. We can find that the inferior-centroid may become the potential centroid. Therefore, we have reason to put both centroids and inferior-centroids into centroid set in the step "Centroids Selection", which means the clusters are all split into two clusters since inferior-centroids are regarded as centroids equally.

Now we have guaranteed that elements of different classes are assigned into different clusters. However, every "One-Class-Cluster" is also split into two clusters, that causes objects of the same classes may be also assigned into different classes. Since Clusters split from "One-Class-Clusters" usually have high similarity to each other and Clusters split from "Two-Class-Clusters" have low similarity to each other (The two clusters split from a "Two-Class-Clusters" each contain one class of objects). We then set a threshold λ to merge clusters split from "One-Class-Clusters" (See Step 6). λ should be set relatively high so that clusters split from "One-Class-Clusters" can be merged back into one cluster while clusters split from "Two-Class-Clusters"remain separate. After the clustering made by DBCAPSIC, we finally get clusters that represent each class of user objects.

3.5 Anomaly Detection

Through DBCAPSIC, we can get several user object clusters, each of which contains similar users, and we also get the center user object of each cluster. Because we think the *BPS* of the center user object represent the *BPSs* of user objects in the same cluster, we extract the *BPS* of the center user object from the cluster and use it as the privilege pattern (*PP*) for anomaly detection.

Definition 10 For a class of user objects, the 2-tuple consisting of a database user set and the behaviour pattern set(*BPS*) of the center user object of this class is defined as the privilege pattern (*PP*), i.e. $\forall cluster_i \in clusterSet$, correspondingly we have $cUO_i \in cUOS$, cUO_i is the center user object of $cluster_i$, thus the privilege pattern (*PP*) of $cluster_i$ is $PP_i = \{US_i, BPS_i\}$,where $US_i = \{UO.user \mid UO \in cluster_i\}$ and $BPS_i = cUO_i.BPS$.

We embed DBCAPSIC into anomaly detection. DBCAPSIC is adopted to make cluster analysis of database user objects (*UO*s) and mine out the privilege patterns (PP) for each class of users from the clusters. With these *PP*s, we can determine whether the operations under detection is anomalous or not.

The description of anomaly detection algorithm is shown in Algorithm 2.

Algorithm 2. Anomaly Detection Algorithm

1.Input:

HR ---Normal operational behaviour records collected in a long history period,

R ---Current operational behaviour records for anomaly detection.

2.*UOS* Construction.

2.1 Initialize the user object set, i.e. $UOS = \varnothing$.

2.2 Traverse the operational behaviour records in HR .

2.2.1 As for the ith record $HRitem_i$, extract the user account name $user$, the database object $object$, and the behaviour type $type$.

2.2.2 If $\exists UO \in UOS$, $UO = \{user, BOS\}$, then turn to Step 2.2.3; else generate a new user object, that is $newUO = \{user, newBOS\}$, where $newBOS = \{newBO\}$, $newBO = \{object, type, frequency\}$ and $frequency = 1$.

Then add $newUO$ to UOS , i.e. $UOS = UOS \cup \{newUO\}$., and turn to Step 2.2.4.

2.2.3 If $\exists BO \in BOS$, $BO = \{object, type, frequency\}$, then update its $frequency$, i.e. $BO.frequency = BO.frequency + 1$; else generate a new behaviour object, i.e. $newBO = \{object, type, frequency\}$, where $frequency = 1$.

Then add $newBO$ to BOS , i.e. $BOS = BOS \cup \{newBO\}$, and turn to Step 2.2.4.

2.2.4 If there still exists some record not traversed, then let $i = i + 1$ and return to step 2.2.1; else the traversal has finished and we get the user object set UOS .

3.*PPS* Construction.

3.1 Use DBCAPSIC to make cluster analysis to get $clusterSet$ and $cUOS$.

3.2 $\forall cluster_i \in clusterSet$, correspondingly we have $cUO_i \in cUOS$, and cUO_i is the center user object of $cluster_i$ .With $cluster_i$ and cUO_i we can construct the privilege pattern PP_i , thus we get the privilege pattern set PPS , i.e. $PPS = \{PP_1, PP_2, ..., PP_r\}$, where $r = |clusterSet|$.

4.Anomaly Detection.

$\forall Ritem \in R$,extract the account name $user_R$, the database object $object_R$ and the behaviour type $type_R$.

4.1 If $\exists PP \in PPS$, $user_R \in PP.US$, then we need another judgment: If $\exists BP \in PP.BPS$, $BP = \{object_R, type_R\}$, then $Ritem$ is marked as normal; else $Ritem$ is marked as anomalous.

4.2 Else $Ritem$ is marked as anomalous.

---

4 Experiment and Analysis

4.1 Evaluation Setting

To evaluate the performance of anomaly detection, detection rate and false alarm rate are usually adopted as two evaluation indexes.

Let N denote the number of operational behaviour records detected in a period, I denote the total number of anomalous behaviour records among the whole records, C denote the number of behaviour records that are considered anomalous, and M is the number of anomalous behaviour records that are neglected.

Definition 11 The detection rate is defined as

$$\eta = \frac{I - M}{I} \times 100\% .$$

Definition 12 The false alarm rate is defined as

$$\mu = \frac{C + M - I}{N} \times 100\% .$$

From the definition we know that the higher the detection rate is and the lower the false alarm rate is, the more perfect the detection result is.

We target at the database of a simulated business system and collect the operational behaviour records with the database audit system (DAS) developed in our laboratory, The DAS capture the data packets and resolve from the packets the database user account name, the operational behaviour type, the database object, the source IP, the operational time and other useful information, which forms the operational behaviour records and can be used for intrusion detection.

We select 11 main tables as the database objects,that is

$$\left\{ \begin{matrix} customer, company, discount, product, kind, sales, \\ sales\_item, shopcart, delivery, goodback, warehouse \end{matrix} \right\} .$$

The operational behaviour types are

$$\{INSERT, DELETE, UPDATE, SELECT, DROP, TRUNCATE\} .$$

The database users has 7 main classes:

Admin, Manager,SalesMan,StoreKeeper, TallyMan, HighCustomer, LowCustomer.

After the collection and process of the normal operations in a period, we comprise a data set with the capacity of 2000000.

The Anomaly Detection is based on the privilege pattern set (PPS) extracted from $clusterSet$ and $cUOS$ output from the DBCAPSIC.

In this paper the anomalous behaviours are divided into 3 types:

(1) An unknown user (illegal user) makes a database operation.

(2) A legal user makes a database operation where the operational behaviour type is within its privilege pattern but the operational object is out of its privilege pattern,

(3) A legal user makes a database operation where the operational object is within its privilege pattern but the behaviour type is out of its privilege pattern.

To validate the anomaly detection algorithm, we need to construct a record set that contains a large proportion of normal records and a small proportion of anomalous records. The record set serves as R in Algorithm 2 and we would like to find whether the anomaly detection algorithm is able to recognize the anomalous records in the set.

The description of the anomalous records construction is shown in Algorithm 3.

Algorithm 3: Anomalous Records Construction Algorithm

1. Input:

PPS --- The privilege pattern set extracted from $clusterSet$ and $cUOS$;

$HRSet$ ---The set of normal operational records collected by the database audit system in a period.

2. Initialize the anomalous record set, i.e. $R_{anomalous} = \varnothing$.

3. $\forall HRitem \in HRSet$, extract from $HRitem$ the account name $user$, the database object $object$ and the operational behaviour type $type$. Traverse PPS and find the $PP \in PPS$ that meets $user \in PP.US$. Then randomly choose one way from Step 3.1, Step 3.2 and Step 3.3 to make transformation of $HRitem$.

3.1 Select or construct an account name $user_{anomalous}$ that meets $user_{anomalous} \notin PP.US$, then construct the new behaviour record $Ritem$ with $user_{anomalous}$, $object$, and $type$.

3.2 Select or construct a database object $object_{anomalous,}$ that meets $\neg \exists BO \in PP.BOS, BO = \{object_{anomalous}, type\}$, then construct the new behaviour record $Ritem$ with $user$, $object_{anomalous}$, and $type$.

3.3 Select or construct a behaviour type $type_{anomalous,}$ that meets $\neg \exists BO \in PP.BOS, BO = \{object, type_{anomalous}\}$, then construct the new behaviour record $HRitem$ with $user$, $object$, and $type_{anomalous}$.

4. $\forall HRitem \in HRSet$,we have made a transformation according to Step 3.1, Step 3.2 or Step 3.3 and get n anomalous records correspondingly. Add the n anomalous records to $R_{anomalous}$ thus we get the anomalous record set.

5. Output;

The anomalous record set $R_{anomalous}$.

With Algorithm 3, we generate some anomalous records and mix them with many normal records, constructing several record set with the capacity of 1000. We adopt the anomaly detection algorithm (Algorithm 2) to validate whether it can detect the anomaly record correctly.

4.2 Result and Analysis

In DBCAPSIC we set the merge coefficient $\lambda = 0.8$ and 7 clusters are generated by DBCAPSIC. Each cluster has one privilege pattern, 7 privilege patterns mined out from the 7 clusters respectively compromises the privilege pattern set.[1]

We generate some anomalous records based on Algorithm 3 and mix them with normal records, constructing several record sets with the capacity of 1000. We adopt the anomaly detection algorithm to validate whether it can detect the anomaly record correctly.

The result of the anomaly detection is shown in Tab.1.

From Tab.1 we can find that based on the privilege patterns extracted from clusters generated through DBCAPSIC we can detect the anomalous behaviour records effectively. The detection rate reaches 100% but there are still some false alarmed records(that is, some normal behaviour records are mistaken as anomalous ones). This is because we use the center user object's behaviour pattern set as the privilege pattern of the cluster it belongs to; but sometimes the center user object's behaviour patterns cannot cover all normal behaviour patterns of users in the cluster. It is because the center user does not make such operations or the operations has not been captured by the database audit system (the packet loss of the database audit system is not discussed here). Therefore, some normal operations will be mistaken it as anomalous ones.

Table 1. Experimental Result of Anomaly Detection

No.	Capacity	A.R.	D.R.	F.R.	I.R.	η	μ
1	1000	25	30	5	0	100%	0.50%
2	1000	25	26	1	0	100%	0.10%
3	1000	30	32	2	0	100%	0.20%
4	1000	30	30	0	0	100%	0.00%
5	1000	30	33	3	0	100%	0.30%

A.R.---The number of anomalous records.
D.R.---The number of records detected as anomalous ones.
F.R.---The number of records detected falsely as anomalous ones.
I.R.--- The number of anomalous records neglected.

However, if a normal operational behaviour is false alarmed,it can be corrected by human review, whereas if an anomalous behaviour is neglected, it may cause unexpected hazards afterwards, thus the neglected anomalous behaviour is more terrible than the false alarmed normal behaviour. It is plausible to eliminate neglected anomalous records at the expense of a little increase of false alarm rate.

[1] We cannot present the mined privilege patterns due to space limit, please refer to http://yunpan.cn/cw9LYX9FnTPLr (with the password :d7f3) for the full version of this paper.

5 Conclusion

Anomaly detection is an important aspect in database security and is attracting more and more attention recently. We adopt cluster analysis techniques in anomaly detection and propose a novel clustering algorithm called DBCAPSIC. With DBCAPSIC, we can mine out the privilege patterns for different classes of users from massive history operational records, and then we can detect the real-time operations made by various types of database users and discover the anomalous operations among them. The simulation experiment shows a relatively good performance of our method. This is of enormous practical value since it enriches methods for the database audit system in anomaly detection and improves the adaptability.of the database audit system under unsupervised conditions to discover intrusion behaviours. More effort is needed in future study to improve the representation of privilege pattern for each cluster so that the false alarm rate can be further reduced.

Acknowledgments. We thanks for the support from Nuclear Takamoto Significant Special and National Development and Reform Commission Information Security Special. We also thanks for the careful reviews and valuable suggestions from the anonymous reviewers.

References

1. Anderson, J.P.: Computer security threat monitoring and surveillance. Technical report, James P. Anderson Company, Fort Washington, Pennsylvania (1980)
2. Lee, W., Stolfo, S.J.: Data mining approaches for intrusion detection. In: Usenix Security (1998)
3. Denning, D.E.: An intrusion-detection model. IEEE Transactions on Software Engineering **2**, 222–232 (1987)
4. Sherif, J.S., Dearmond, T.G.: Intrusion detection: systems and models. In: 2012 IEEE 21st International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, pp. 115–115. IEEE Computer Society (2002)
5. Eskin, E., Miller, M., Zhong, Z.D., et al.: Adaptive model generation for intrusion detection systems (2000)
6. Ashoor, A.S., Gore, S.: Intrusion detection system (IDS): case study. In: Proceedings of 2011 International Conference on Advanced Materials Engineering (ICAME 2011) (2011)
7. Kokane, S., Jadhav, A., Mandhare, N., et al.: Intrusion Detection in RBAC Model
8. Zhang, J., Chen, X.: Research on Intrusion Detection of Database based on Rough Set. Physics Procedia **25**, 1637–1641 (2012)
9. Zhang, Y., Ye, X., Xie, F., et al.: A practical database intrusion detection system framework. In: Ninth IEEE International Conference on Computer and Information Technology, CIT 2009, vol. 1, pp. 342–347. IEEE (2009)
10. Pang-Ning, T., Steinbach, M., Kumar, V.: Introduction to data mining. Library of Congress (2006)
11. Campos, M.M., Milenova, B.L.: Creation and deployment of data mining-based intrusion detection systems in oracle database l0g. In: Proceedings of the Fourth International Conference on Machine Learning and Applications, 2005, p. 8. IEEE (2005)

12. Bloedorn, E., Christiansen, A.D., Hill, W., et al.: Data mining for network intrusion detection: How to get started. MITRE Technical Report (2001)
13. Feng, W., Zhang, Q., Hu, G., et al.: Mining network data for intrusion detection through combining SVMs with ant colony networks. Future Generation Computer Systems **37**, 127–140 (2014)
14. Kim, M.Y., Lee, D.H.: Data-mining based SQL injection attack detection using internal query trees. Expert Systems with Applications **41**(11), 5416–5430 (2014)
15. Pietraszek, T., Tanner, A.: Data mining and machine learning—towards reducing false positives in intrusion detection. Information Security Technical Report **10**(3), 169–183 (2005)
16. Khan, S.S., Ahmad, A.: Cluster center initialization algorithm for K-means clustering. Pattern Recognition Letters **25**(11), 1293–1302 (2004)
17. Mitra, P., Murthy, C.A., Pal, S.K.: Density-based multiscale data condensation. IEEE Transactions on Pattern Analysis and Machine Intelligence **24**(6), 734–747 (2002)
18. Macqueen, J., et al.: Some methods for classification and analysis of multivariate observations. In: Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, pp. 281–297 1967: Smith, T.F., Waterman, M.S.: Identification of Common Molecular Subsequences. J. Mol. Biol. **147**, 195–197 (1981)
19. Brossette, S.E., Ahymel, P.: Data mining and infection control. Clinics in Laboratory Medicine **28**(1) (2008)
20. Giudici, P.: Applied Data Mining: Statistical Methods for Business and Industry. Journal of the American Statistical Association **38**(475), 1317–1318 (2006)
21. Luan, J.: Data Mining and Knowledge Management in Higher Education -Potential Applications. Cluster Analysis (2002)
22. Zou, B., Ma, X., Kemme, B., Newton, G., Precup, D.: Data mining using relational database management systems. In: Ng, W.-K., Kitsuregawa, M., Li, J., Chang, K. (eds.) PAKDD 2006. LNCS (LNAI), vol. 3918, pp. 657–667. Springer, Heidelberg (2006)