Fast Detection of Database System Abuse Behaviors Based on Data Mining Approach

Yubao Liu
Sun Yat-Sen University
Department of Computer Science of Sun Yat-Sen University
Guangzhou, China, 510275
liuyubao@mail.sysu.edu.cn

Jiarong Cai
Sun Yat-Sen University
Department of Computer Science of Sun Yat-Sen University
Guangzhou, China, 510275
kelvin2004_cai@163.com

Zhilan Huang
Sun Yat-Sen University
Department of Computer Science of Sun Yat-Sen University
Guangzhou, China, 510275
santahzl@gmail.com

Jingwen Yu
Sun Yat-Sen University
Department of Computer Science of Sun Yat-Sen University
Guangzhou, China, 510275
yjw831@163.com

Jian Yin
Sun Yat-Sen University
Department of Computer Science of Sun Yat-Sen University
Guangzhou, China, 510275
issjyin@mail.sysu.edu.cn

ABSTRACT
Recently, the mining of system log datasets has be widely used in the system security application field such as the detection of abuse behaviors. At present, most of efforts concentrate on the network or operating system level. There are few works concentrated on database system application. In this paper, we present the concept of access profile to represent the user behavior characteristics of accessing database system and study the problem of mining maximal access profiles for fast detection of database system insider abuse behaviors by legitimate users. Based on the existing FP-tree structure, a new mining algorithm MMAP is presented for our problem. A new constraint of relation distance, which is based on the foreign key dependencies of relations, is also presented to reduce the mining algorithm search space. An anomaly-based detection model is build based on MMAP algorithm for performance experiments. The experimental results show that our approach works efficiently for detecting the abuse behaviors of database system.

Categories and Subject Descriptors
H.2.8 [**Database Applications**]: Data Mining

General Terms
Algorithms, Management, Performance, Security

INFOSCALE 2007, June 6-8, Suzhou, China

DOI 10.4108/infoscale.2007.916

Keywords
Data System Abuse Behaviors, Access Profiles, Maximal Access Profiles, FP-tree, Anomaly-based Detection Model

1. INTRODUCTION
Recently, the mining of system log datasets has be widely used in the system security application field such as the detection of abuse behaviors. At present, most of efforts concentrate on the network or operating system level [1-4][19][20]. There are few works concentrated on database system application.

For database system, the auditing mechanisms are designed to record all system activities in great details and ensure that no intrusion evidence will be missed. So database system audit data is a kind of high-speed and high-volume data. The high-speed and high volume data requires the run-time execution of mining models be very efficient. The long delay in data analysis can not satisfy such run-time execution.

In this paper, we present the concept of access profile to represent the user behavior characteristics of accessing database system (the definition is in section 2) and study the problem of mining maximal access profiles for fast detection of insider abuse by legitimate users. Based on the existing FP-tree structure [6], a new mining algorithm called MMAP for the mining of maximal access profiles is presented. An anomaly-based detection model is build based on the proposed MMAP algorithm for performance experiments. The experimental results show that our approach works efficiently for detecting the abuse behaviors of database system.

There are some related works to our problem. The DEMIS system [5] is also to detect the abuse user behaviors of database system. However, in DEMIS, the user profiles are defined as the set of attributes level of relation. Audit data build on such fine granularity will reduce the whole database system performance. Our behavior characteristics are defined on the higher data granularity such as relation. In addition, DEMIS uses SQL-based queries to implement the mining of user profiles for achieving the tight integration with database system, but that also reduces the detection speed in a certain extent. The existing ADMIT system [2] also presents some efficient solution for fast detection of real time application. However ADMIT system is based on the clustering of command-level data of operating system.

The mining of maximal frequent itemsets is also related to our work [7][8][13][14]. Different from these algorithms, our MMAP can further efficiently reduce the algorithm search space by the constraint of database objects contained in the access profiles. Among these algorithms, the Max algorithm [7] uses the depth-first strategy to search the maximal frequent itemsets and also is a fast algorithm and used in our experiments.

In addition, the traditional mining of frequent itemsets and association rules are also related to our work. For example, the generalized association rules mining [9][6], the constrained mining of frequent itemsets [11][12][15][16][18], top-k frequent itemsets [17] and integration of association rules and database management system [10] etc.

The rest paper is organized as following: the definitions of basic concepts are in section 2, the mining algorithms are in section 3, the experiment results are in section 4, and the conclusions are in section 5.

2. THE DEFINITIONS OF BASIC CONCEPTS

For database system, the user's access characteristics can be generalized as: *who* accesses *what* objects from *where* at *when*. For example, "Tom (i.e. *who*) reads some records from relation R_1, R_2 (i.e. *what*) at 10:30 (i.e. *when*) from IP address: 222.111.222.1 (i.e. *where*)." These objects often are recorded in each database session of system audit dataset.

We present a new concept of *access profile* to represent the user's behavior characteristics. An access profile is a set of the combination of such characteristic objects (i.e. *who*, *what*, *where* and *when*). For example, the access profile {Tom, 222.111.222.1} represents the user 'Tom' accesses database system from the address '222.111.222.1'.

Definition 1 (access profile): An access profile AP is a set of the combination of the values of characteristic objects *who, what, where* and *when* of database audit system.

Example1. Assume that r and w denote 'read' and 'write' operations respectively. Then the following combinations are access profiles: AP_1 = {Tom, r(R_1), w(R_2), 202.110.0.1, 12:30}, AP_2 = {r(R_1), w(R_2))}, AP_3 = {202.111.0.1, 12:30}, AP_4={Jack, r(R_2)}.

Definition 2 (frequent access profile): Given an access profile AP of a session database, we say AP is frequent if support(AP)≥ min-support, where support(AP) is the number of database sessions in which all the objects in AP are contained together. The support threshold constraint, min-support, is generally specified in advance.

Definition 3 (maximal access profile): Given a set of frequent access profile APS, we say $x \subseteq$ APS is a maximal frequent access profile if there is no other frequent access profile $y \subseteq$ APS such that $x \neq y$ and $y \subset x$.

Example2. Assume that APS = {AP_1, AP_2, AP_3, AP_4} is a set of frequent access profiles, where AP_1, AP_2, AP_3, AP_4 are same to those in example 1. Then the maximal frequent access profiles are AP_1 and AP_4 since they are not contained in the other frequent access profiles.

Definition 4 (the maximal access profiles mining): Given a session database SD, the maximal access profiles mining is to mine all the maximal access profiles from SD.

Property 1. If x is a frequent access profile, then any subset of x is also frequent.

Proof. If x is a frequent access profile, then we have support(x)≥ min-support. For any set x'⊆ x, x is an access profile according to the definition of access profile. The amount of database sessions containing all the objects in x' is not less than that of database sessions containing all the objects in x. So we have support(x') ≥ min-support and x' is also frequent.

Property 2. If x is a non-frequent access profile, then any superset of x is also non-frequent.

Proof. If x is not a frequent access profile, then we have support(x)< min-support. For any set x'⊇ x, x' is also an access profile. The amount of database sessions containing all the objects in x' is not larger than that of database sessions containing all the objects in x. We have support(x') ≤ support(x) < min-support. So x' is not frequent.

Property 3. If x is a maximal frequent access profile, then any subset of x is also frequent.

Proof. It is easy to know from the definition of maximal frequent access profiles and property 1.

From property 3, it is known that any frequent access profiles are contained in the maximal access profiles (i.e. upward closed). So, it is sufficient to discover only all maximal frequent access profiles but all the frequent access profiles while detecting such abuse behaviors.

3. THE MINING ALGORITHMS

3.1 The Pre-processing

The goal of pre-processing is to generate the session database from the original system audit data.

First, we omit some information unrelated to access profiles, such as the CPU spending etc. According to the attribute of connection id of database system audit logs, we group the same connections into the same database sessions. Notice that the values of time stamp are generalized in the pre-processing. For example, the value of "10:30" is generalized as "am" and the value of "17:40" is generalized as "pm".

The purpose of generation is to avoid generating redundant access profiles. For example, the time stamps "9:30" and "9:31" have the near time stamp and represent almost the same meanings, that is, the database is accessed about 9:30. After the generalization, there possibly exist some duplicated records and then the redundant records are removed in the pre-processing.

Each session is associated with "session id" and "access profiles". Table 1 shows an example of session database generated by the pre-processing in which u_i, h_i ($1 \le i \le 2$) denotes the coded database users and host respectively.

Table 1. An example of session database

Session ID	Access Profiles
S_1	am, u_1, $r(R_2)$, $w(R_2)$, h_1
S_2	am, u_2, $r(R_2)$, $w(R_1)$, h_2
S_3	pm, u_1, $w(R_1)$, h_1

3.2 The FP-Tree of Session Database

An FP-tree (frequent pattern tree) is a variation of the *trie* data structure, which is a prefix-tree structure for storing crucial and compressed information about frequent patterns.

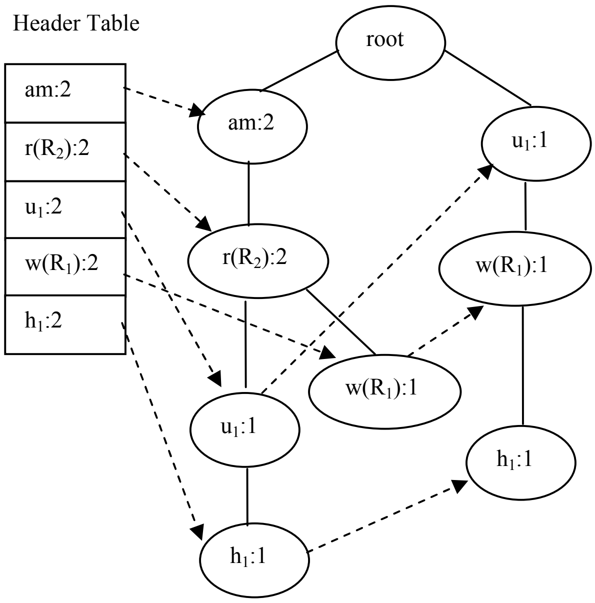

Figure 1. An FP-tree of the session databases in table 1, where the min-support =2.

It consists of one root, a set of item prefix subtrees as the children of the root, and a frequent item header table. Each node in the item prefix subtree consists of four fields: *item-name*, *count*, *node-link* and *parent-link*, where *item-name* indicates which item this node represents, *count* indicates the number of transactions containing items in the portion of the path reaching this node, *node-link* links to the next node in the FP-tree carrying the same item-name, or null if there is none, and *parent-link* links to the parent of this node.

Each entry in the frequent item header table consists of three fields: *item-name*, *item-count* and *head of node-link*. The *head of node-link* points to the first node in the FP-tree carrying the item-name. It is noticed that the items in header table is sorted in the descending order of support count. The FP-tree structure is an efficient data structure for constructing the mining algorithm of frequent itemset without generating any candidate itemsets. In our study, we directly call the construction procedure of FP-tree for building the FP-tree of the session database.

The FP-tree for an transaction database can be constructed in the following steps: [6]

(a) Scan the transaction database *D* once. Collect the set of frequent items *F* and their supports. Sort *F* in support descending order as *L*, the list of frequent items.

(b) Create the root of an FP-tree *T*. For each transaction *Trans* in D do the following. Select and sort the frequent items in each transaction according to the order of *L*. Let the sorted frequent itemset list in *Trans* be [*p*|*P*], where *p* is the first element and *P* is the remaining list. Call *insert-tree*([*p*|*P*], *T*), which is performed as follows. If *T* has a child *N* such that *N*.item-name = *p*.item-name, then increment *N*'s count by 1; else create a new node *N*, and let its count be 1, its parent link be linked to *T*, and *P* is nonempty, call *insert-tree*([*p*|*P*], *T*) recursively.

Similarly, for our session database, we can also construct an FP-tree. An example of FP-tree of the session database in table1 is given in fig.1.

3.3 The MMAP Algorithm

The MMAP algorithm is constructed through the modifications of the existing FP-Growth algorithm.

```
Algorithm MMAP
Inputs: The FP-tree of session database T, the minimal
support threshold min-support
Outputs: The maximal frequent access profile set MAP
Methods:
1. begin
2.     MAP=∅;
3.     Call the procedure MFP(T, MAP, null);
4.     return MAP;
5. end.
```

Figure 2. The description of MMAP algorithm

According to the definition of maximal access profile and the construction of FP-tree, we observe that the maximal access profiles only exist in the sets of frequent access profiles which are generated through the conditional pattern bases (a 'subdatabase' which consists of the set of prefix paths in the FP-tree co-occurring with the suffix pattern) of FP-tree. So we can directly output the maximal access profiles but generating all the combination of different objects in a single path of FP-tree, which is done in FP-Growth algorithm to generate all the frequent patterns.

The description of MMAP algorithm is given in fig.2. The inputs of MMAP are the FP-tree of session database and the minimal support threshold constraint. The maximal frequent access profile set MAP is the output.

The procedure MFP in MMAP is used to discover all the maximal access profiles whose last objects is x and put the mined maximal access profiles in the set MAP. The description of MFP is given in fig.3.

Procedure MFP (T, MAP, x)
1. begin
2. if T only contains a single path P then
3. $m=\{a_1 \cup a_2 \cup \dots \cup a_n | a_i \in P\} \cup x$;
4. m.support=a_n.support;
5. if m is not a subset of the sets in MAP then
6. MAP=MAP$\cup$m;
7. Delete the subsets of m from MAP;
8. end
9. else
10. for each $a_i \in$Htable of T do
11. $h=a_i \cup x$;
12. h.support=a_i.support;
13. Constructing conditional FP-tree of h, T_h, with min-support and the conditional patterns bases of h;
14. If $T_h \neq$null then
15. MFP(T_h, MAP, h);
16. end
17. end.

Figure 3. The description of MFP algorithm

Let's take an example to show the process of MMAP algorithm. Provided that the min-support is equal to 2 and the FP-tree is shown in fig.1.

Firstly MAP is set to $\varnothing$ and the MFP procedure MFP(root, $\varnothing$, null) is called. Due to the FP-tree of root does not only contain a single path, then the tenth line of MFP procedure is executed. Then a_i is set to 'h_1' and h= $a_i \cup x = h_1 \cup null = h_1$. The conditional pattern bases of h_1 are: {am:1, r(R_2):1, u_1:1}, {u_1:1, w(R_1):1}. The conditional FP-tree of T_{h1} is constructed and it only contains a single path (i.e.{u_1:2}, and the other objects whose supports are less than the min-supports are removed in the T_{h1}) in which only the object 'u_1' is included.

Then the procedure MFP(T_{h1}, MAP, h_1) is recursively called. The second line of MFP procedure is executed and the objects 'u_1' and 'h_1'are combined into a maximal access profiles, that is, {u_1, h_1}. Then the procedure MFP recursively return.

Similarly, the rest objects of header table are chosen to generate the maximal access profiles and the final MAP={{u_1, h_1}, {am, r(R_2)}}.

3.4 Algorithm Complexity Discussion

The MMAP is based on the existing FP-Growth algorithm and has the same algorithm framework with FP-Growth. So the correctness and completeness of MMAP is same to FP-Growth algorithm [6]. Due to MMAP only generates the maximal access profiles and removes lots of frequent access profiles, the MMAP algorithm is more fast.

In addition, we can also further improve our algorithm by reducing the algorithm search space using the relation distance constraint.

3.5 The Relation Distance Constraint

Compared to the general itemsets, the access profiles consist of the special database semantics, which can be used to further reduce the algorithm search space. In detail, we use the relation distance constraint, which is based on the database semantics, to reduce the algorithm search space.

The relation distance, which is based on the foreign key dependencies of relations, represents the close degree of relations.

Definition 5 (relation graph): All the relations R = {R_1, R_2, …, R_n}in a database system are viewed as the graph nodes. For any two relations R_i, $R_j \in$ R, if they are related through foreign key dependencies, then there is one connected edge between nodes R_i and R_j in the relation graph otherwise they are not connected.

Definition 6 (relation distance): Given a set of relations R = {R_1, R_2, …, R_n}, the relation distance rd(R_i, R_j) is defined as following:

(1) $$rd(R_i,R_j) = \frac{\text{shortestdist}(R_i, R_j)}{\text{Max}\{\text{shortestdist}(R_p, R_q) | R_p, R_q \in R\}}$$, where shortestdist(R_i, R_j) means the shortest path between the nodes R_i and R_j in the relation graph.

(2) rd(R_i,R_j) is undefined if R_i and R_j are not connected in relation graph.

(3) In particular, for a given relation set R, we define rd(R) = Max{rd(R_i, R_j)| R_i, $R_j \in$R and $1 \le i, j \le n$}, and rd(R)=0 if R only contains a single relation or R is $\varnothing$.

We normalize the distance measure by the maximum shortest distance between any pair of relations in the database so that the value of distance measure falls in the range of 0 to 1.

According to the definition of relation distance, for the relation graph in fig.4, we have rd(R_1, R_2)= rd(R_2, R_1)=1/2=0.5, rd(R_1, R_3) =rd(R_3, R_1)=2/2=1, rd(R_1, R_4)= rd(R_4, R_1)=1/2=0.5 and rd(R_3, R_4)= rd(R_3, R_4)=2/2=1. Notice that the distance of two relations is smaller, and then they are closer and have higher possibility of being referenced together in a database session.

Due to all the relations in a real system are not totally connected with each other, there possibly exist multi-relation graphs. For each relation graph, we call the Dijkstra's algorithm to compute the shortest paths of any two nodes.

The computation of relation distances can be done in our pre-processing and the computed relation distance can be stored as a kind of background knowledge constraint. By specifying the relation distance constraint, the mining algorithm further reduces the search space and focuses on the interesting maximal access profiles.

The mining algorithm with such constraint is easily constructed. In detail, we just prune *what* characteristic objects containing such relations whose distances are not satisfied with the specified constraint from header table. There is no need to modify the other parts of MMAP algorithm.

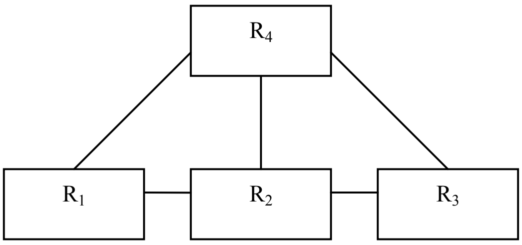

Figure 4. An example of relation graph in which four relations R_1, R_2, R_3 and R_4 are included.

4. THE EXPERIMENT RESULTS

An anomaly-based detection model is build for the performance experiments. This model consists of two steps, that is, the learning of training dataset and detecting of testing dataset. In this model, two mining algorithms are implemented for performance comparison. One is our MMAP algorithm, and the other is Max algorithm [13] that is also a fast mining algorithm. The tested performance includes learning runtimes, detecting runtimes, true positive rate (TPR) and false positive rate (FPR).

All experiments are conducted on a PC platform with an AMD 1.6G CPU, 240M RAM, 40G Hard disk and Windows XP OS. The detection model is implemented using Java 1.4.

Based on the audit system of Microsoft SQL server 2000, we generate two database session logs through our pre-processing. The first log is normal session database and consists of 1,000,000 session transactions. The second log is abuse session database and consists of 400,000 session transactions. The maximal number of characteristic objects of each session transaction is set as 50. The total number of relations is set as 50. In order to increase the reliability of test results, the whole process is repeated for 10 times and the average results are rounded and then reported.

In the first set of experiments, the learning runtimes with different min-supports are tested. The learning runtimes of both algorithms are consumed to generate the normal access profiles are tested. The min-supports are varied from 5% to 20% with an interval 5% and the size of training dataset is set as 1,000,000 transactions. The results are given in fig.5.

In the second set of experiments, the detecting runtimes with different min-supports are tested. The detecting runtimes of both algorithms are consumed to generate the current access profiles and check if the current access profiles are matched with the normal access profiles. We say a normal access profile AP_1 and a current access profile AP_2 are matched iff $AP_2 \subseteq AP_1$. In this set of experiments, the set of normal access profiles is the one mined in the first set of experiments with min-support 5%. The size of the testing dataset is fixed as 100,000 transactions. The results are given in fig.6.

In the third and fourth set of experiments, the learning and detecting runtimes with different sizes of datasets (DS) are tested. In the third set of experiments, the sizes of training dataset are varied from 100,000 to 400,000 with an interval 100,000. The min-support is fixed as 5%. The results are given in fig.7. In the fourth set of experiments, the detecting runtimes with different sizes of testing datasets are tested. The sizes of testing datasets are varied from 100,000 to 400,000 with an interval 100,000. The min-support is fixed as 5%. The results are given in fig.8.

From the results of fig.5-fig.8, we can see that: (1) The learning runtimes and detecting runtimes of MMAP obviously outperforms Max in which the relation distance is not used. It is because the relation distance constraint can efficiently reduce the MMAP algorithm search space. (2) Both runtimes are decreased with the increased min-supports. It is because more maximal access profiles are filtered with higher min-supports.

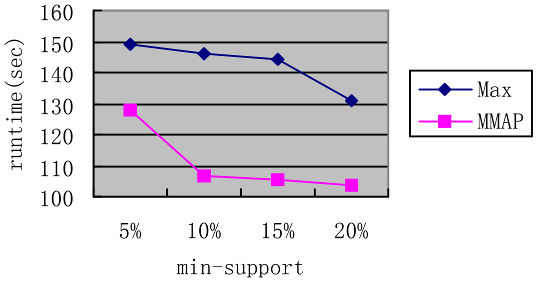

Figure 5. The learning runtimes vs SP.

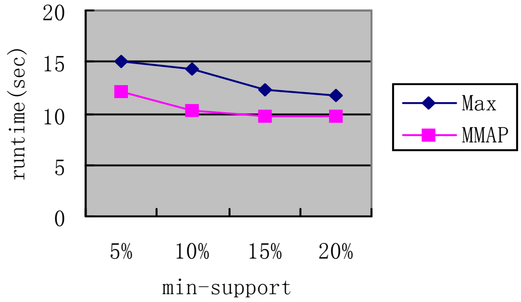

Figure 6. The detecting runtimes vs SP.

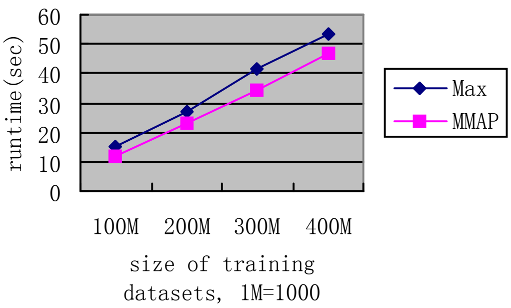

Figure 7. The learning runtimes vs DS.

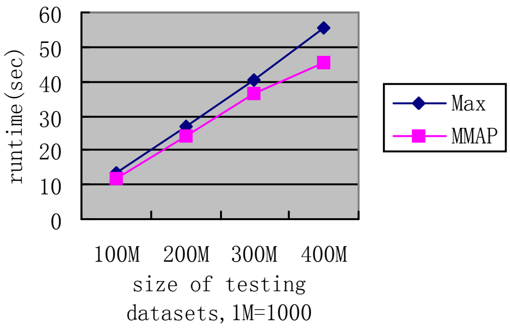

Figure 8. The detecting runtimes vs DS.

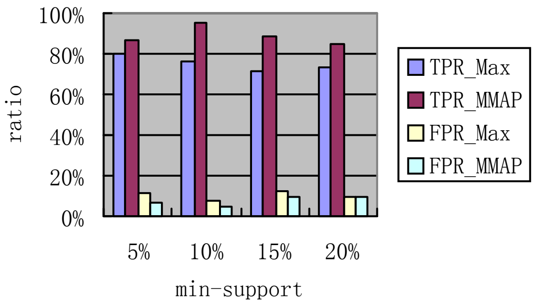

Figure 9. The true/fase positive rate vs min-support.

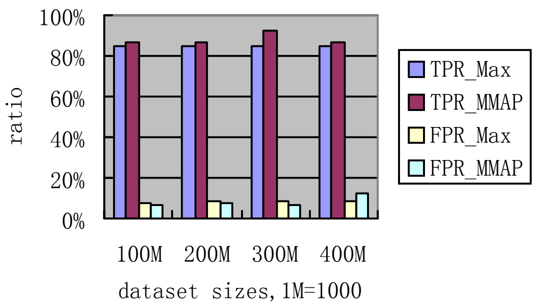

Figure 10. The true/fase positive rate vs sizes of testing datasets.

In the fifth set of experiments, we test the TPR and FPR with different min-supports. Both the sizes of testing and training dataset are 100,000 transactions. The min-supports are varied from 10% to 25% with an interval 5%. The results are given in fig.9.

In the sixth set of experiments, we test the TPR and FPR with different sizes of testing datasets. The sizes of datasets are varied from 100,000 to 400,000 with an interval 100,000. The min-supports are fixed as 5%. The results are given in fig.10.

From the results of fig.9 and fig.10, we can see that MMAP outperforms Max in the aspects of TPR and FPR. We also find that TPR is more sensitive to the min-support but the size of testing dataset. It is increased with the increase of min-supports, whereas it is steady with different sizes of testing datasets. Though the changing of FPR is relatively small, it is seen that the FPR is also more sensitive to the min-support but the size of testing dataset. Actually, MMAP algorithm generates more maximal access profiles with smaller min-supports, and that can provide finer comparison about the current and the normal access profiles for TPR and FPR.

5. CONCLUSIONS

Recently, the mining of system log datasets has be widely used in the system security application field such as the detection of abuse behaviors. At present, most of efforts concentrate on the network or operating system level. There are few works concentrated on database system application.

For database system, the auditing mechanisms are designed to record all system activities in great details and ensure that no intrusion evidence will be missed. So database system audit data is a kind of high-speed and high-volume data. The high-speed and high volume data requires the run-time execution of mining models be very efficient. The long delay in data analysis can not satisfy such run-time execution.

In this paper, we present the concept of access profiles to represent the user behavior characteristics of accessing database system and study the problem of mining maximal access profiles for fast detection of database system insider abuse behaviors. Based on the existing FP-tree structure, a new mining algorithm called MMAP is presented. A new constraint of relation distance is also presented to reduce the mining algorithm search space. The experimental results based on the anomaly model show that our approach works efficiently for detecting the abuse behaviors.

In future work, we plant to integrate the anomaly model into a data mining tool which will be used for the analysis of real database system abuse behaviors.

6. ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for the helpful comments.

This work was supported by the National Natural Science Foundation of China (60573097), Natural Science Foundation of Guangdong Province (05200302, 06104916), Research Foundation of Science and Technology Plan Project in Guangdong Province (2005B10101032), Research Foundation of Disciplines Leading to Doctorate degree of Chinese Universities (20050558017), and Program for New Century Excellent Talents in University of China.

7. REFERENCES

[1] Lee, W., Fan, W. Mining System Audit Data: Opportunities and Challenges. SIGMOD Record 4 (2001), pp.35-44.

[2] Sequeria, K., Zaki, M. ADMIT: Anomaly-based Data Mining for Intrusions. In: Proc. KDD 2002, pp.386-395.

[3] Wang, K., Stolfo, J.S. Anomalous Payload-based Network Intrusion Detection. In: Proc. RAID 2004, pp.203-222.

[4] Mahoney, M., Chan, P. K. Learning Nonstationary Models of Normal Network Traffic for Detecting Novel Attacks. In: Proc. KDD 2002, pp.376-385.

[5] Chung, Y.C., Gertz, M., Levitt, N.K. DEMIDS: A Misuse Detection System for Database Systems. In: Proc. IFIP IICIS 1999. pp.159-178.

[6] Han,J., Pei, J., and Yin, Y. Mining Frequent Patterns without Candidate Generation. In: Proc. SIGMOD 2000, pp.1-12.

[7] Grahne G, Zhu, JF. High performance mining of maximal frequent itemsets. In: Proc. SIAM Workshop on High Performance Data Mining (HPDM 2003), pp.135-143.

[8] Burdick, D., Calimlim, M., Gehrke, J. Mafia. A maximal frequent itemset algorithm for transactional databases. In: Proc. ICDE2001, pp.443-452.

[9] Rakesh Agrawal, Ramakrishnan Srikant Fast Algorithms for Mining Association Rules in Large Databases. In Proc. VLDB 1994, pp.487-499.

[10] Sunita Sarawagi, Shiby Thomas, Rakesh Agrawal. Integrating Mining with Relational Database Systems: Alternatives and Implications. In Proc. SIGMOD Conference 1998, pp.343-354.

[11] Roberto J. Bayardo Jr., Rakesh Agrawal, Dimitrios Gunopulos. Constraint-Based Rule Mining in Large, Dense Databases. In Proc. ICDE 1999, pp.188-197.

[12] Ke Wang, Yu He, Jiawei Han. Pushing Support Constraints Into Association Rules Mining. IEEE Trans. Knowl. Data Eng. 15(3): 642-658 (2003).

[13] Bayardo R. Efficiently mining long patterns from databases. In: Haas LM, ed. Proc. of the ACM SIGMOD Int’l Conf. on Management of Data. New York: ACM Press, 1998. 85−93.

[14] Gouda K, Zaki MJ. Efficiently mining maximal frequent itemsets. In: Proc. of the 1st IEEE Int’l Conf. on Data Mining. 2001. 163−170.

[15] Jian Pei, Jiawei Han, Laks V. S. Lakshmanan. Pushing Convertible Constraints in Frequent Itemset Mining. Data Min. Knowl. Discov. 8(3): 227-252 (2004).

[16] Yin-Ling Cheung, Ada Wai-Chee Fu. Mining Frequent Itemsets without Support Threshold: With and without Item Constraints. IEEE Trans. Knowl. Data Eng. 16(9): 1052-1069 (2004).

[17] Raymond Chi-Wing Wong, Ada Wai-Chee Fu. Mining top-K frequent itemsets from data streams. Data Min. Knowl. Discov. 13(2): 193-217 (2006).

[18] Ke Wang, Yu He, Jiawei Han. Mining Frequent Itemsets Using Support Constraints. In Proc. VLDB 2000, pp.43-52.

[19] Ke Wang, Gabriela Cretu, Salvatore J. Stolfo. Anomalous Payload-Based Worm Detection and Signature Generation. In Proc. RAID 2005, pp.227-246.

[20] Jian Pei, Shambhu J. Upadhyaya, Faisal Farooq, Venugopal Govindaraju. Data Mining for Intrusion Detection: Techniques, Applications and Systems. In Proc. ICDE 2004, pp. 877.