The ISC Int'l Journal of Information Security
ISeCure

July 2014, Volume 6, Number 2 (pp. 155–167)
http://www.isecure-journal.org

A Hybrid Approach for Database Intrusion Detection at Transaction and Inter-transaction Levels

Mostafa Doroudian [1], and Hamid Reza Shahriari [1,*]
[1] *Computer Engineering and Information Technology Department, Amirkabir University of Technology, Tehran, Iran*

A R T I C L E I N F O.

Article history:
Received: 19 March 2014
Revised: 13 November 2014
Accepted: 16 November 2014
Published Online: 24 November 2014

Keywords:
Intrusion Detection, Database Security, State machine, Inter-Transaction Dependency, Inter-Transaction Sequence.

A B S T R A C T

Nowadays, information plays an important role in organizations. Sensitive information is often stored in databases. Traditional mechanisms such as encryption, access control, and authentication cannot provide a high level of confidence. Therefore, the existence of Intrusion Detection Systems in databases are necessary. In this paper, we propose an intrusion detection system for detecting attacks in both database transaction level and inter-transaction level (user task level). For this purpose, we propose a detection method at transaction level, which is based on describing the expected transactions within the database applications. Then at inter-transaction level, we propose a detection method that is based on anomaly detection and uses data mining to find dependency and sequence rules. The main advantage of this system, in comparison with the previous database intrusion detection systems, is that it can detect malicious behaviors in both transaction and inter-transaction levels. Also, it gains advantages of a hybrid method, including specification-based detection and anomaly detection, to minimize both false positive and false negative alarms. In order to evaluate the accuracy of the proposed system, some experiments have been done. The experiment results demonstrate that the true positive rate (*recall* metric) is higher than 80%, and the false positive rate is lower than 10% per different data sets and choosing appropriate ranges for support and confidence thresholds. The experimental evaluation results show high accuracy and effectiveness of the proposed system.

1 Introduction

Many organizations employ DBMS as the main technology of data management in order to store and access their data. This information is often considered as a valuable asset in the organizations. Hence, the data in the databases must be protected from unauthorized access and changes. Traditional database security mechanisms such as encryption, access control, and authentication are not sufficient for protecting of sensitive information against innovative security attacks [1]. Intrusion detection systems (IDSs) in the database can be used for detecting attacks whenever traditional prevention mechanisms are infeasible or may be bypassed.

The user task in this paper refers to a set of transactions that are always submitted to the DBMS together to achieve a certain goal, also the time gap

* Corresponding author.
Email addresses: doroodian@aut.ac.ir (M. Doroudian), shahriari@aut.ac.ir (H.R. Shahriari).
ISSN: 2008-2045 © 2014 ISC. All rights reserved.

between two tasks of one user is much longer than it between two transactions within a user task. Here, the database attacks refer to malicious behaviors that may appear in the requests issued to database systems. It is possible that all of the transactions in a user task are legitimate, whereas there exist some anomalies in relations among transactions. We can divide the detectable attacks by database IDSs into four categories including SQL query level attack, the transaction level, inter-transaction level (user task level) as well as SQL injection attack.

The existing database intrusion detection methods are often based on anomaly detection approach, and anomaly-based methods are usually prone to generating a relatively large number of false alarms. On the other hand, most of these proposed methods focus on detecting the database attacks in SQL query or transaction levels and cannot detect anomalies among transactions at the user task level.

In this paper, we propose an intrusion detection system for database in transaction level and user task level. The proposed system at transaction level uses a detection method that is based on specifying the expected transactions within database applications in the organization. The issued transactions that are different from the existing specifications are considered as malicious transactions. At the user task level, we present a type of detection method, which is based on anomaly detection. This method creates normal behavior profile via mining the dependency and sequence relation rules among transactions. The differences between new user tasks and existing dependency and sequence rules are recognized as abnormal events.

The main advantage of our approach is that we use a hybrid method at transaction and user task levels, including specification-based detection and anomaly detection. Thus we gain their both advantages to minimize false positive and false negative detection alarms.

The rest of the paper is organized as follows. Section 2 describes past efforts related to database IDS. In Section 3, we explain our proposed approach. Section 4 presents the evaluation results of the presented approach. Section 5 offers conclusions of the paper.

2 Related Work

There exist a few research works that concentrate on database IDSs. These systems are used to detect malicious behaviors in one of three levels; SQL query, transaction and user task. Also, SQL injection attacks are attacks that can be detected by some of these systems.

One of the early works in this field is a database misuse detection system called DEMIDS [1]. It uses a data mining algorithm to capture working scopes of users. The algorithm extracts frequent itemsets as working scopes using the notion of distance measures. The frequent itemsets are stored in user profiles to describe typical behaviors of users. The techniques offered in [2, 3] can be used in detecting of query level attacks. They apply the profiling of normal SQL query for a database role using a Nave Bayes clustering approach on the training log file. The anomalous behaviors of users are identified by comparing SQL queries submitted by users and normal SQL queries stored in role profiles.

Most of the previous researches in this field concentrate on modeling the normal behaviors of users at transaction level to detect malicious activities against database system. The method used in [4] applies the same approach existing in [2, 3] (mentioned above) for detecting attacks at transaction level. Hu *et al.* [5] propose an intrusion detection method based on mining the dependencies among data items at transaction level using static analysis of the application source code. These dependencies are represented as sets of reading and writing operations for each data item. Another intrusion detection method presented by Hu *et al.* [6] uses a sequential pattern mining technique on the training database log for identifying frequent sequences among data items at transaction level. The transactions that do not comply with the data dependencies are recognized as malicious transactions. The proposed method [7] is similar to [6] with the difference that modifies an existing sequential mining algorithm and make it security sensitive sequential mining by introducing weights for each attribute based on the sensitivity group. Lee *et al.* [8] describe an intrusion detection method in real-time database systems using time signatures. In this method the temporal data objects are tagged with time signatures of update latency ranges that are unknown to the intruders.

Intrusions at inter-transaction level have been less considered by existing database IDSs. Hu *et al.* [9] present an algorithm for identifying frequent data dependencies among data items at the inter-transaction level in order to detect user task level attacks. In this approach, first, transactions existing in the training database log are clustered into user tasks using a proposed algorithm. Then, the data dependencies in user task level are discovered using a proposed algorithm. These dependencies are represented as inter-transaction dependency rules and inter-transaction sequence rules for each data item.

There are some other systems that have been specially presented for detecting SQL injection attacks. For instance, the technique used by DIDAFIT [10] tries to characterize legitimate accesses through fingerprinting their constitutive SQL queries to identify anomalous accesses to the database system. This technique also imposed a constraint on the ordering of SQL queries submitted. The proposed system in [11] is also a novel framework based on anomaly detection that creates a fingerprint of an application program based on SQL queries. Then, it uses association rule mining techniques on the fingerprints to extract useful rules that represent the normal behaviors of database applications.

3 Proposed Approach

Our proposed approach is an intrusion detection mechanism that can be employed for detecting the malicious requests issued to a database system by a set of database applications within an organization, although this mechanism can be used for only one application instead of a group of database applications at organization granularity level. Here, the assumption is that all the considered applications use a finite set of transactions for issuing requests to the database system and an identifier is assigned to each transaction. Also, the transactions in a user task can be submitted from different applications.

For identifying the malicious behaviors at submitted transactions to the database, the proposed IDS first identifies the existing transactions within organization applications through analyzing the source code of applications. Then, the system specifies these transactions through a formal specification method. In the detection phase, the new transactions existing in the current database log are recognized as legitimate if they are matched to the corresponding stored specifications.

In the training phase, the proposed system discovers the dependency and sequence rules among transactions in the user tasks through a data mining algorithm in order to identify the abnormal relations among transactions in performing a user task. For this purpose, the system uses a training database log that contains the normal database requests of considered applications during a certain period of time. In the detection phase, if transactions appear in a new user task so that the relations among them are matched to the stored rules, this user task is considered as normal.

3.1 System Architecture

The proposed system acts as a security mechanism that is independent of the database system operations and placed beside the DBMS to enhance its security. For that, in detection phase, the proposed system uses the current database log file to analyze the requests submitted from the database applications for detecting the malicious behaviors at transactions and the relations among them.

The architecture of proposed system consists of two modules that include the *Analysis and Generation* module along with the *Detection* module. The system architecture is shown in Figure 1.

The *Analysis and Generation* module consists of two sub-modules of *Specification Creation* and *Rule Generation*. The *Specification Creation* sub-module first identifies the transactions existing in organization applications using the *Static Analyzer* and then the identified transactions are specified as a type of state machine specifications at *Specification Creator*. The *Rule Preprocessor* at *Rule Generation* sub-module clusters the transactions existing in the training database log into user tasks and sends them to *Rule Generator*, and the *Rule Generator* generates the dependency and sequence rules through a data mining algorithm

For every new transaction and new user task in database log, the *Detection Preprocessor* extracts the required information from them and sends extracted information to *Specification Matcher* and *Rule Matcher* respectively. The new transactions are matched against the existing specifications based on their information at *Specification Matcher*. The *Rule Matcher* examines the relations among transactions in the performed user tasks based on the stored rules. If existing transactions in a new user task and relations among them are recognized as legitimate, then the user task is considered as normal and stored into *Normal UTs* storage, otherwise, the system raises an alarm. The *Rule Generator* periodically uses existing user tasks in *Normal UTs* storage to update the existing rules.

3.2 Analysis and Generation Module

As mentioned earlier, the *Specification Creation* sub-module of the *Analysis and Generation* module specifies the identified transactions as state machines. Whereas, the *Rule Generation* sub-module discovers the dependency and sequence rules of relations among identified transactions and then stores them.

3.2.1 Static Analyzer

The *Static Analyzer* analyzes the source code of database applications in the organization and places extracted information from identifying transactions into a file named *Specification*. This analyzer places derived information in the *Specification* file as follows:

- The "T_b" expression is inserted at the beginning of each transaction and "T_e" expression is placed at the end of it with a space character of the

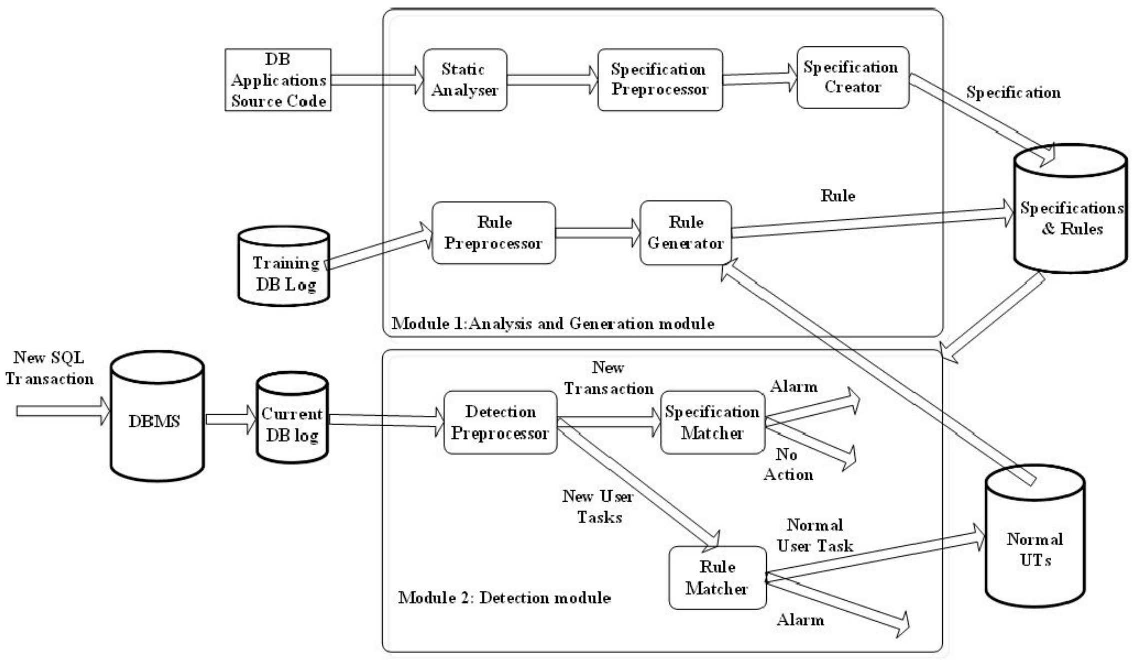

Figure 1. Architecture of proposed system

previous expression.

- It inserts the transaction identifier (*Tr*) after "T_b" expression at the beginning of each transaction.
- The SQL queries existing at transactions are placed into the file with a "*?*" symbol between adjacent SQL queries.

Furthermore, some methods are considered for showing the special statuses such as conditional choosing among several SQL queries and a loop on the SQL query sequence.

3.2.2 Extracting Transaction Information

The *Specification Preprocessor* extracts the required information from existing SQL queries within the *Specification* file and then transfers them to *Specification Creator* in the form of tuples. This component considers each SQL query containing the *where* clause as two queries. The first query is equivalent to the *projection* clause of intended query, which consists of SQL commands such as *Select*, *Insert*, *Update*, and *Delete* on the set of attribute within the database relations. Also, the second query is equivalent to a *Select* query on the relation attributes existing at the *where* clause. The reason of this conversion is that the existing attributes at the *where* clause are read before applying the equality operator. In addition, the command related to the *where* clause is processed at first.

This component creates a tuple for each SQL query and then puts the derived information from query in it, which each tuple consists of the following nine fields:

(*Beg_Tr*, *Cmd*, *Count*, *Att [ ]*, *WC*, *End_Tr*, *Tr_ID*, *Additional Fields*, *Tuple_ID*)

The *Beg_Tr* is a binary (bit) field that contains 1 when the intended SQL query is at the beginning of the transaction. The *Cmd* field shows the involved SQL command that has values 'S', 'I', 'U', and 'D' whenever the SQL command is equal to *Select*, *Insert*, *Update*, and *Delete* respectively. The *Count* field corresponds to the number of the database relation attributes included in the SQL query command. The *Att [ ]* vector contains the attributes number included in the SQL command respectively. The value of *WC* binary field is equal to 1 if the SQL command is derived from the where clause. The *End_Tr* is a binary field that contains 1 when the SQL query is at the end of the transaction. The *Tr_ID* field corresponds to the identifier of transaction containing the expected SQL command. The *Additional Fields* are fields that used in the particular cases such as conditional choosing among the SQL commands and a loop of SQL queries. Furthermore, the sequence number of the tuple is stored in the *Tuple_ID* field. By the way, this component processes the transactions with the same identifier only once.

3.2.3 Transaction Specification

As mentioned earlier, the *Specification Creator* creates a state machine specification for each expected transaction, which is based on the information derived from the SQL queries within the intended transaction. Here, we use a type of state machine that is called *Ex-*

tended Finite State Automata (*EFSA*) [12]. In general, an EFSM is a tuple (Σ, Q, s, f, V, D, δ), where:

- Σ corresponds to *event alphabet* of the EFSA, where each event is characterized by an event name along with the event arguments.
- Q is a finite set of states of the EFSA.
- $s \in Q$ describes the start state of the EFSA.
- $f \in Q$ corresponds to the *final state*.
- V is a finite tuple (v_1, ..., v_n) of *state variables*.
- D indicates a finite tuple (D_1, ..., D_n) of domain values for variables.
- δ : $Q \times D \times \Sigma \rightarrow (Q, D)$ denotes the transition relations between states.

The transition relation is characterized as rules of the form:

$$event(x_1, ..., x_n)|condition \longrightarrow action$$

Where, *event* is an event name, and the variables x_1, ..., x_n represent the arguments of this event. The *condition* is a condition expression that if satisfied, the actions defined by the action expression will be done. An example of single state transition is depicted in Figure 2.

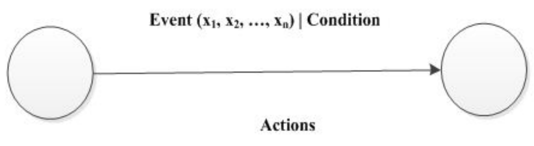

Figure 2. Example of single state transition

This component creates the state transitions according to the incoming tuples from the *Preprocessor* that are associated to the expected SQL commands. The condition expression of each transition examines the matching between the arguments of issued events and the values existing at the state variables. Also, the state variables will be set to expected values of the next SQL query, in the action expression.

Here, the state machine uses two groups of state variables that the first group with the name of *query* variables contains information about the involved SQL queries. The second group of variables that are called *transaction* variables includes information about the transactions and also about the relations among SQL queries. Moreover, it applies two types of events with the names *query* and *abort*. The *query* event consists of two arguments (*q*, *TrID*), where *q* contains the information about the issued SQL query, which corresponds to the tuple fields created by *Preprocessor*. The *TrID* is string and corresponds to the identifier of the issued transaction. The *abort* event consists of the *TrID* argument and indicates the occurrence of the *rollback* command within the intended transaction.

Moreover, the transaction variables are *CurrSt*, *CurrTr*, and *NextSt*. The *CurrTr* variable contains the identifier of the current transaction, which is initialized in the first transition of each state machine. The *CurrSt* variable corresponds to the current state number, also the *NextSt* variable contains the states numbers that are reachable from the current state.

For example, consider the following transaction:

T_1: Select Name, Family, Number from Employee
Update Table1 set Name = Joe, Family = "Petre"

Figure 3 shows the state machine for above transaction, where the numbers 1, 2, 3, and 4 are assigned to attributes Name, Family, Number, and Tel, respectively.

3.2.4 Clustering Transactions into User Tasks

The *Rule Preprocessor* component uses a training audit log file to extract information about existing transactions in database applications. For this purpose, the component first extracts some information from each transaction, such as start time, end time and identifier, and stores them into three separate vectors. Then, the three mentioned vectors are used for clustering the transactions into user tasks using the clustering method in [9]. As mentioned earlier, a user task is a set of transactions that are submitted to the DBMS together to perform a given task of one user. Generally the time gap between two adjacent tasks of one user is much bigger than the gap between two adjacent transactions inside one user task. The mentioned clustering algorithm uses this observation for clustering transactions to different user tasks. After doing this, the component sends sequence of transaction identifiers separately for each identified user task to *Rule Generator*.

3.2.5 Inter-transaction Rules Generation

The *Rule Generator* is used for discovering the inter-transaction dependency rules and inter-transaction sequence rules that are defined as follows.

Definition 1: An ***Inter-Transaction Dependency*** is the set of transactions that are almost relevant to each other so that often issued together in a database user task, formally is the set of the form {T_1, T_2, ..., T_n}, n $\geq$ 2, where T_1, ..., T_n are instance transactions belonging to the set T that consists of the identifier of all the transactions identified in the database applications of the organization. Meanwhile, the order of elements in the inter-transaction dependency is not

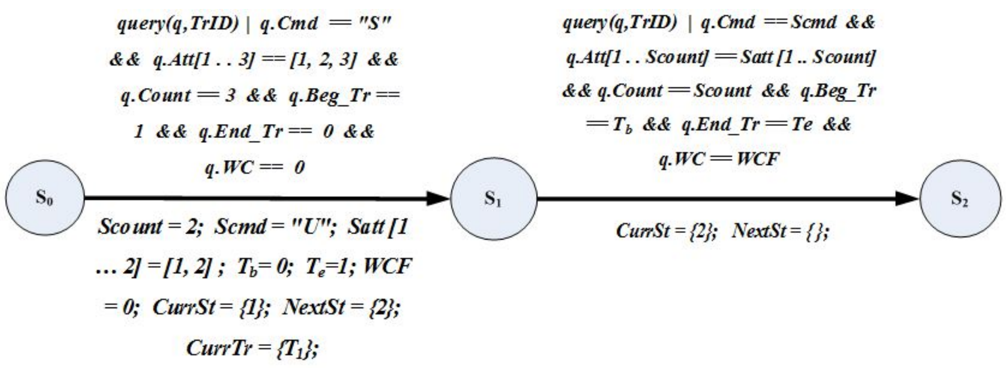
query(q,TrID) | q.Cmd == "S" && q.Att[1 . . 3] == [1, 2, 3] && q.Count == 3 && q.Beg_Tr == 1 && q.End_Tr == 0 && q.WC == 0
S0
S1
S2
Scount = 2; Scmd = "U"; Satt [1 … 2] = [1, 2] ; Tb = 0; Te = 1; WCF = 0; CurrSt = {1}; NextSt = {2}; CurrTr = {T1};
query(q,TrID) | q.Cmd == Scmd && q.Att[1 . . Scount] == Satt [1 .. Scount] && q.Count == Scount && q.Beg_Tr == Tb && q.End_Tr == Te && q.WC == WCF
CurrSt = {2}; NextSt = { };

Figure 3. Example of state machine specification for transaction T_1

important. The support value for an inter-transaction dependency is the percentage of the identified user tasks within the training database log that contain this dependency among the intended transactions. An inter-transaction dependency is considered to be a frequent dependency, if calculated support value for this dependency is above the predefined support threshold. All the frequent inter-transaction dependencies are stored in the set *ITD* to be used for generating the inter-transaction dependency rules. The general idea of Appriori algorithm [13] can be used for discovering the frequent inter-transaction dependencies.

The Algorithm 1 presents the procedure for generating frequent inter-transaction dependencies. The input to this algorithm is the sequences of transaction identifiers existing in the identified user tasks and identified transactions set T along with support s. Also, the maximum number of transactions in a user task is placed as input to this algorithm. In this algorithm, the symbol '$\|$' is used for showing the concatenation operator of two sequences.

Definition 2: An ***Inter-Transaction Dependency Rule*** with support value s and confidence value c is defined as a rule of the form:

$$\{T_1, T_2, \ldots, T_j\} \longrightarrow \{T_{j+1}, T_{j+2}, \ldots, T_n\}\ [s,\ c],\ n \geq 2$$

Where $T_1, \ldots, T_n$ correspond to the instance transactions belonging to the set T that consists of all the specified transactions existing in the organization. This rule represents that if transactions $T_1, \ldots, T_j$ are submitted through a user task to the database system, the transactions $T_{j+1}, \ldots, T_n$ is most probably issued to the database within this user task. The rules are generated based on the frequent inter-transaction dependencies that already defined. The support value s of a rule corresponds to the percentage of the identified user tasks within the training database log that this rule is generated based on them. Also, the confidence value c of a rule is the percentage of the identified user tasks that contain both the antecedent and consequent of the rule. For generating the inter-transaction dependency rules, we first extract a set *C_ITDR* of rules as candidate rules from the discovered frequent dependencies existing in the set *ITD* and calculate the confidence value for each of them. Algorithm 2 contains steps for generating inter-transaction dependency rules. The sequences of transaction identifiers within identified user tasks and identified transactions set T are placed as input to the algorithm.

Definition 3: An **Inter-Transaction Sequence** consists of the transactions that are often accessed in sequence within a user task of the database application. Formally, an inter-transaction sequence is represented as $< T_1, T_2, \ldots, T_n >$, $n \geq 2$, where $T_1, \ldots$, T_n correspond to the instance transactions belonging to the set T that contains the specified transactions. If support value of an inter-transaction sequence is above the predefined support threshold, the sequence is selected as a frequent sequence and stored into the set *ITS* to be used for generating the inter-transaction sequence rules. We can use the sequential pattern mining algorithm AprioriAll in [14] for discovering the frequent inter-transaction sequences.

Definition 4: An ***Inter-Transaction Sequence Rule*** is a rule with one of two forms:

$$< T_1, T_2, \ldots, T_j > \longrightarrow < T_{j+1}, T_{j+2}, \ldots, T_n > [s,\ c],\ n \geq 2$$

$$< T_1, T_2, \ldots, T_j > \longleftarrow < T_{j+1}, T_{j+2}, \ldots, T_n > [s,\ c],\ n \geq 2$$

Where $T_1, \ldots, T_n$ are instance transactions belonging to the set T that consists of the identifier of all the existing transactions in the organization. The first rule determines that

If transactions $T_1, \ldots, T_j$ are issued to the database system with the mentioned order among them within a user task, the transaction sequence $< T_{j+1}, \ldots, T_n >$ is accessed within this user task afterward, with support text s and confidence c. The second rule represents that if transactions $T_{j+1}, \ldots, T_n$ are submitted to the database system as a sequence through a user task, the transactions $T_1, \ldots, T_j$ are accessed in se-

Algorithm 1 Inter-Transaction Dependency Generation Algorithm

Input: The set UTs consisting of sequences of transaction identifiers existing in legitimate identified user tasks, identified transaction set T, support threshold s, maximum number of transactions in user task with input argument n.

Output: The frequent inter-transaction dependency set ITD.

```
1: Initialize inter-transaction dependency set $ITD$, i.e., $ITD = \{\}$
2: for j = 1,...,n do
3:    $L_j = \{\}, c_j = \{\}$
4: end for
5: for each transaction $T_i \in T$ do
6:    if Support($T_i$) > s then
7:       $L_1 = L_1 \cup \{T_i\}$
8:    else
9:       if Support($T_i$)=0 then
10:         Delete stored specification for transaction $T_i$
11:      end if
12:   end if
13: end for
14: i =2
15: for each $itd \in L_{i-1}$ and frequent transaction $ft \in L_1$ do
16:    $c_i = c_i \cup \{itd||ft\}$
17: end for
18: for each $itd_c \in c_i$ do
19:    if Support($itd_c$) > s then
20:       $L_i = L_i \cup \{itd_c\}$
21:    end if
22: end for
23: if $L_i \neq \{\}$ then
24:    $i = i + 1$
25:    Jump to step 15
26: else
27:    for j=2,..., i-1 do
28:       for each $itd \in L_j$ do
29:          $ITD = ITD \cup \{itd\}$
30:       end for
31:    end for
32: end if
```

quence before them within this user task, with support s and confidence c. The inter-transaction sequence rules are generated using the same approach of generating the inter-transaction dependency rules with the difference that the frequent inter-transaction sequences are used for generating them. The generated inter-transaction sequence rules are placed into the set *ITSR*.

Algorithm 3 illustrates the algorithm for generating inter-transaction sequence rules. The input to this algorithm is the sequences of transaction identifiers in the identified user tasks and identified transactions set T along with confidence c. Also, the generated dependency rules set *ITDR* is placed as input to this algorithm to modify based on the generated sequence rules. In this algorithm, the $ITDR_{T_1,T_2,\ldots,T_j}$ is used for showing the set of inter-transaction dependency rules that their antecedents consist of transactions $T_1, T_2, \ldots, T_j$. Also, the $ITSR_{T_1,T_2,\ldots,T_j}$ set is a set of inter-transaction sequence rules that the transaction sequence $< T_1, T_2, \ldots, T_j >$ are placed as antecedent of these rules.

3.3 Detection Module

The *Detection* module identifies the malicious transactions and the abnormal user tasks based on the existing transaction specifications and existing inter-transaction rules respectively. For this purpose, the *Detection* module examines the SQL queries issued by database applications in the form of transactions and user tasks stored in the database log file. In this module, the *Detection Preprocessor* immediately after observing each new issued transaction in the log file, extracts its information and sends this information to

Algorithm 2 Inter-Transaction Dependency Rule Generation Algorithm

Input: The set UTs consisting of the sequences of transaction identifiers in legitimate identified user tasks, identified transactions set T, confidence threshold c.

Output: Inter-transaction dependency rules set $ITDR$.

```
1:  Initialize inter-transaction dependency set $ITD$, i.e., $ITD = \{\}$
2:  Initialize inter-transaction dependency rules set $ITDR$, i.e., $ITDR = \{\}$
3:  Generate frequent inter-transaction dependencies using the frequent itemsets mining algorithm Apriori [13]
    from the set $UTs$ and store them into the set $ITD$.
4:  for each frequent inter-transaction dependency $itd \in ITD$ do
5:    for each set $d \subseteq itd$ do
6:      if $\frac{Support(itd)}{Support(d)} > c$ then
7:        $ITDR = ITDR \cup \{d \longrightarrow itd - d\}$
8:      end if
9:    end for
10: end for
11: for each rule $r \in ITDR$ do
12:   if there exists a rule $r^* \in ITDR$ that its antecedent is equal to $r$'s antecedent and $r$'s consequence is the
      subset of its consequence then
13:     $ITDR = ITDR - \{r\}$
14:   end if
15: end for
```

the *Specification Matcher* through the tuples of the form introduced in *Specification Preprocessor* component (Section 3.2.2). Also after identifying any user task consisting of legitimate transactions, the component forwards sequence of transaction identifiers to the *Rule Matcher*.

The *Specification Matcher* is used for matching the new transactions against the existing specifications corresponding to them. To do so, the component uses the transaction information received from the *Preprocessor* as tuples and the stored specification as a state machine corresponding to the transaction identifier. For each SQL query within the transaction, a state transition must be done in the intended state machine. The transaction is considered as a legitimate user transaction if at the end of matching operation, the state machine is in its final state.

The *Rule Matcher* matches the user tasks against the existing inter-transaction rules. For that, this component uses sequence of transaction identifiers within new user tasks. In order to perform the matching operation, the component first selects the rules that their antecedent have been appeared in the user task. Then, the selected rules are grouped into the sets based on their antecedent so that each set contains the rules with the same antecedent. If the user task sequence satisfies at least one rule from each set, the user task is considered to be a normal user task.

Generally, detection process consists of three following steps:

- **Step 1**: the *Detection Preprocessor* performs following actions respectively:
  - ○ Extract transaction information after observing each new issued transaction in the current log file.
  - ○ Send this information to the *Specification Matcher*.
  - ○ Forward sequence of transaction identifiers to the *Rule Matcher* after identifying any user task consisting of legitimate transactions.
- **Step 2**: the *Specification Matcher* matches each new issued transaction against corresponding specification and raises an alarm after observing incoherence.
- **Step 3**: the *Rule Matcher* matches the user tasks against the existing inter-transaction rules.

The algorithm for detecting malicious behaviors at user task level is shown in Algorithm 4. The sequence of transaction identifiers in each new user task is placed as input to the algorithm.

4 Evaluations and Results

We analyze our proposed method in terms of qualitative and experimental. At the first part, we present the results of several experiments that have been conducted to evaluate the accuracy rate of proposed method. At the second part, we compare our method with the previous methods in qualitative terms.

4.1 Experimental Evaluation

Several experiments have been conducted for evaluating the effectiveness of the proposed database intru-

Algorithm 3 Inter-transaction Sequence Rule Generation Algorithm

Input: The set UTs consisting of sequences of transaction identifiers existing in the legitimate user tasks, generated inter-transaction dependency rules set $ITDR$, identified transaction set T, confidence threshold c.
Output: Inter-transaction sequences set $ITSR$, modified inter-transaction dependency rules $ITDR$.

```
1:  Initialize frequent inter-transaction sequences set $ITS$, i.e., $ITS = \{\}$
2:  Initialize inter-transaction sequence rule set $ITSR$, i.e., $ITSR = \{\}$
3:  Generate frequent inter-transaction sequences using sequential pattern mining algorithm AprioriAll [14] from
    the set $UT$s and store them into the set $ITS$.
4:  for each sequence $its = \langle T_1, T_2, \ldots, T_n \rangle \in ITS$ do
5:    for j =1,..., n-1 do
6:      for each two subsequences $A = \langle T_1, T_2, \ldots, T_j \rangle, B = \langle T_{j+1}, T_{j+2}, \ldots, T_n \rangle$ do
7:        if (Support(⟨ A,B ⟩) /Support(A)) > c then
8:          $ITSR = ITSR \cup \{A \rightarrow B\}$
9:        end if
10:       if (Support(⟨ A,B ⟩) /Support(B)) > c then
11:         $ITSR = ITSR \cup \{A \leftarrow B\}$
12:       end if
13:     end for
14:   end for
15: end for
16: for each rule $r \in ITDR$ do
17:   if there exists a rule $r^* \in ITSR$ that its antecedent and its consequence are equal to $r$'s antecedent and
      $r$'s consequence respectively then
18:     $ITDR = ITDR - \{r\}$
19:   end if
20: end for
21: for each rule $r \in ITSR$ do
22:   if there exists a rule $r^* \in ITSR$ that its antecedent equal to $r$'s antecedent and $r$'s subsequence of its
      consequence then
23:     $ITSR = ITSR - \{r\}$
24:   end if
25: end for
26: for each rule $r = \langle T_1, T_2, \ldots, T_j \rangle \rightarrow \langle T_{j+1}, T_{j+2}, \ldots, T_n \rangle \in ITSR$ or rule $r = \langle T_1, T_2, \ldots, T_j \rangle \leftarrow$
    $\langle T_{j+1}, T_{j+2}, \ldots, T_n \rangle \in ITSR$ do
27:   $ITSR_{T_1,T_2,\ldots,T_j} = ITSR_{T_1,T_2,\ldots,T_j} \cup \{r\}$
28: end for
29: for each rule $r = \langle T_1, T_2, \ldots, T_j \rangle \rightarrow \langle T_{j+1}, T_{j+2}, \ldots, T_n \rangle \in ITDR$ do
30:   $ITDR_{T_1,T_2,\ldots,T_j} = ITDR_{T_1,T_2,\ldots,T_j} \cup \{r\}$
31: end for
32: for each set $s = ITSR_{T_1,T_2,\ldots,T_n} \subseteq ITSR$ do
33:   Assign an identifier to the set $s$
34: end for
35: for each set $s = ITDR_{T_1,T_2,\ldots,T_n} \subseteq ITDR$ do
36:   Assign an identifier to the set $s$
37: end for
```

Algorithm 4 Detection Algorithm at User Task Level

Input: The set UT consisting of the sequence of transaction identifiers in new user task ut, inter-transaction sequence rules set $ITSR$ and identifiers assigned to the subsets of $ITSR$ set, inter-transaction dependency rules set $ITDR$ and identifiers assigned to the subsets of $ITDR$ set.

Output: Status of user task ut ("user task ut is normal" or "user task ut is abnormal").

```
1:  Initialize candidate rules set $C\_R$, i.e., $C\_R = \{\}$
2:  for each set $ITDR_{T_1,T_2,\ldots,T_i} \subseteq ITDR$ with the identifier $k$ do
3:    if the transactions $T_1, T_2, \ldots, T_i$ appear in the transaction sequence of user task $ut$ then
4:      $C\_R = C\_R \cup \{k\}$
5:    end if
6:  end for
7:  for each set $ITSR_{T_1,T_2,\ldots,T_i} \subseteq ITSR$ with the identifier $k$ do
8:    if the transaction sequence $< T_1, T_2, \ldots, T_i >$ appears in the transaction sequence of user task $ut$ then
9:      $C\_R = C\_R \cup \{k\}$
10:   end if
11: end for
12: for each set $R \subseteq ITDR$ or set $R \subseteq ITSR$ with the identifier $k \in C\_R$ do
13:   if there exists a rule $r \in R$ that is not satisfied by $ut$ then
14:     Consider $ut$ as an abnormal user task and raise an alarm
15:   else
16:     Consider $ut$ as a normal user task and store $ut$ into normal $UTs$ storage
17:   end if
18: end for
```

sion detection system. We used the programming language C# from .NET Framework 4.5 for preliminary implementation of proposed system. Also, a training database log file, 10^6 normal user tasks with average 5 legitimate transactions per user tasks, was used in the *Analysis and Generation* module. For generating this database log, we considered an instance organization with 6 database applications. The programming languages of these applications were C# from .NET Framework and Java. Also, these applications were connected to the database systems of Microsoft SQL Server 2012. The database applications contained 45 different types of user tasks and 10 different types of transactions. The transactions are issued by different applications in the form of user tasks (at least 4 transactions per user tasks) so that the same transactions have the same transaction identifier. Also, the existing user tasks at this log were generated randomly based on the certain real scenarios among the database applications in instance organization. In addition, we used four different test data sets as instance database logs in *Detection* module. These data sets consist of a number of normal user tasks along with the several user tasks were treated as malicious user tasks that contained either malicious transactions or abnormal relations among transactions. The number of user tasks existing in these data sets were 500, 1000, 10^4 and 10^5 respectively, and the percentage of anomalous user tasks in data sets were 5%, 10%, 15%, and 20%. Meanwhile, the malicious user tasks in these data sets were very near to the normal user tasks so that there was only one anomalous relation among transactions in a malicious user task, therefore, the conducted evaluations represent the accuracy rate of the proposed system in the worst case.

For evaluating the system, we analyze the results of experiments using the standard measures of *True Positive Rate* (*Recall*), *False Positive Rate*, and *Precision*. These statistics are defined as follows:

$$True\ Positive\ Rate = \frac{\#TruePositive}{\#TruePositive\ +\ \#FalseNegative}$$

$$False\ Positive\ Rate = \frac{\#FalsePositive}{\#FalsePositive\ +\ \#TrueNegative}$$

$$Precision = \frac{\#TruePositive}{\#TruePositive\ +\ \#FalsePositive}$$

Where, #*True Positive* corresponds to the number of intended attacks that the proposed system is able to detect them, and #*True Negative* is defined as the number of genuine events that the proposed system considers them as genuine. Also, #*False Positive* and #*False Negative* are the number of false alarms raised by the system and the number of malicious events identified as genuine, respectively.

In the considered experiments, first, the support threshold was set to 0.1 and the values of true/false positive rate and precision were calculated by setting the confidence threshold to the values from 0.5 to 1. Then, the confidence threshold was considered equal to 0.9 and the values of true/false positive rate and precision were calculated per the various values of the support from 0.05 to 0.5.

Figure 4 illustrates the relationship between the confidence threshold of rules and true positive rate of the proposed system against the four test data sets, and Figure 5 illustrates the relationship between the confidence threshold of rules and false positive rate. Also, Figure 6 shows the ratio of precision measure to confidence threshold of rules. From Figure 4, 5, and 6 it can be seen that the false positive rate and precision are very sensitive to the variations of confidence, while the true positive rate is not very susceptible to these variations.

Figure 7 illustrates the relationship between the support threshold of rules and true positive rate of the proposed system against the four data sets, and Figure 8 illustrates the relationship between the support threshold of rules and false positive rate. Also, Figure 9 shows the ratio of precision measure to support threshold of rules. From Figure 7, 8, and 9, it can be observed that the true positive rate and precision are very susceptible to the changes of support, whereas the false positive rate is not very sensitive to these changes.

By observing the Figure 4, 5, 7, and 8 it is taken that the true positive rate graph at all points has a higher value than the corresponding points in the false positive rate graph; That means the proposed system raises the true alarms much more than the false alarms in all the cases. This implies that the proposed system has high accuracy and effectiveness in detecting database malicious behaviors at both transaction and user task levels.

From the viewable results in Figure 4 to Figure 9 it can be taken the ideal range of confidence threshold corresponds to [0.83, 1] and the desired range of support threshold includes the values from 0.05 to 0.115.

4.2 Qualitative Analysis

We consider the methods [1, 2, 6, 9, 10] in the case of database IDS with the most citations and different approaches to compare them with our method in qualitative terms.

We can divide profile granularity levels to four levels; user, role, application, and organization. Hu *et al.* [6, 9] keep profiles at the level of organization, whereas Chung *et al.* [1] and Bertino *et al.* [2] represent their profiles in the granularity level of user and role respectively. Also, the normal profiles of method [10] are captured in application level. Unlike these methods, our proposed method can be used in both granularity levels of application and organization and also can be easily extended to apply at the granularity level of user or role. It is important that the profile granularity level can be chosen according to the type of organization. However, our method is scalable enough to use in different levels according to the type of organization.

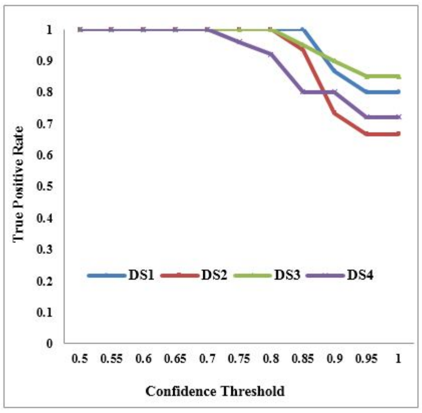

Figure 4. Relation between true positive rate and confidence threshold of rules for four data sets

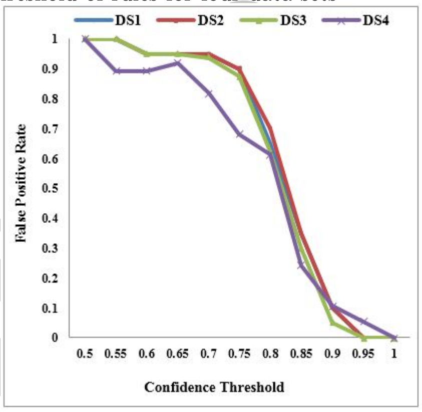

Figure 5. Relation between false positive rate and confidence threshold of rules for four data sets

Our proposed method is capable to identify malicious database behaviors at transaction and inter-transaction levels as well as many types of SQL injection attacks, whereas previous methods are mostly used for identifying attacks only at one of the mentioned levels. For instance, Chung *et al.* [1], Bertino *et al.* [2] and Lee *et al.* [10] propose some methods to identify attacks in SQL query level. Also, Hu *et al.* [6] concentrate on transaction level attacks, whereas the proposed method by Hu *et al.* [9] only deals with anomalous behaviors in relations among transactions.

Furthermore, as mentioned earlier, our method is based on the specification in the levels of query and transaction so that specifies transactional features in a formal manner and precisely. This causes that our method produces false alarms less than other methods that use frequent attribute set (Chung *et al.* [1]), tuple (Bertino *et al.* [2]), attribute dependency (Hu *et al.* [6, 9]), and query fingerprint (Lee *et al.* [10]) to represent a transaction.

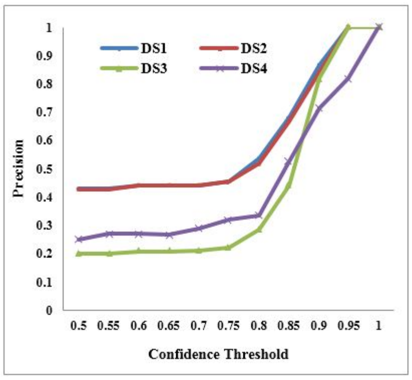

Figure 6. Relation between precision and confidence threshold of rules for four data sets

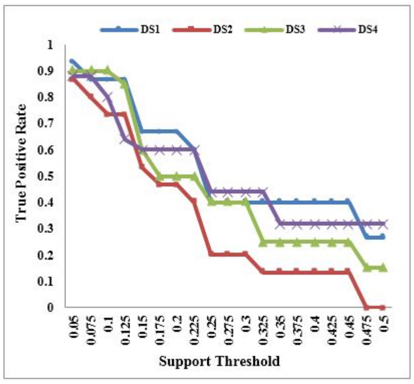

Figure 7. Relation between true positive rate and support threshold of rules for four data sets

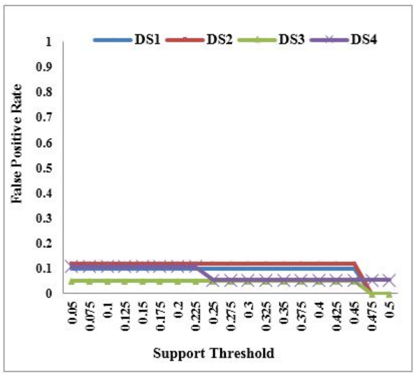

Figure 8. Relation between false positive rate and support threshold of rules for four data sets

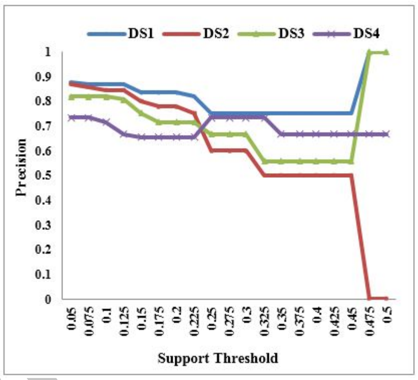

Figure 9. Relation between precision and support threshold of rules for four data sets

5 Conclusions

In this paper, we proposed a novel IDS to detect malicious behaviors at both transaction and inter-transaction (user task) levels in database systems. For this purpose, first, a specification-based method has been offered at the transaction level by which the expected transactions in existing database applications in the organization are detected. These specifications are used to detect malicious transactions, since the malicious transactions cannot be matched to any of the specifications. Then in user task level, an anomaly-based method is presented, which uses a data mining method to discover dependency and sequence rules of the relations among the identified transactions. The discovered rules are used to detect malicious relations among new transactions at the user task level.

The advantage of the proposed system is that it can detect malicious behaviors in both transaction and inter-transaction levels. This system utilizes a hybrid method (specification-based detection and anomaly detection) to decrease both false positive and false negative alarms. In addition, it can be employed at both granularity levels of application and organization. The results of experimental evaluations demonstrate the system operations to be highly accurate at the desired confidence and support levels.

References

[1] C.Y. Chung, M. Gertz, and K. Levitt. DEMIS: A Misuse Detection System for Database Systems. In *Third Annual IFIP TC-11 WG 11.5 Working Conference on Integrity and Internal Control in Information Systems*, pages 159–178, 1999.

[2] E. Bertino, A. Kamra, and E. Terzi. Intrusion Detection in RBAC-administered Databases. In *Proceedings of the 21st Annual Computer Security Applications Conference (ACSAC)*, pages 170–182, 2005.

[3] A. Kamra, E. Bertino, and E. Terzi. Detecting Anomalous Access Patterns in Relational Databases. *The International Journal on Very Large Data Bases*, 17(5):1063–1077, 2008.

[4] U.P. Rao, G.J. Sahani, and D.R. Patel. Machine Learning Proposed Approach for Detecting Database Intrusions in RBAC Enabled Databases. In *The International Conference on Computing Communication and Networking Technologies (ICCCNT)*, pages 1–4, 2010.

[5] Y. Hu and B. Panda. Identification of Malicious Transactions in Database Systems. In *Proceedings of the International Database Engineering and Applications Symposium (IDEAS '03)*, pages 329–335, 2003.

[6] Y. Hu and B. Panda. A Data Mining Approach for Database Intrusion Detection. In *Proceedings of the ACM Symposium on Applied Computing*, pages 711–716, 2004.

[7] A. Srivastava, S. Sural, and A.K. Majumdar. Weighted Intra-transactional Rule Mining for Database Intrusion Detection. In *Proceedings of the Pacific-Asia Knowledge Discovery and Data (PAKDD)*, pages 611–620, 2006.

[8] V.C.S. Lee, J.A. Stankovic, and S.H. Son. Intrusion Detection in Real-time Database Systems Via Time Signatures. In *Proceedings of the 6th IEEE Real Time Technology and Application Symposium (RTAS)*, pages 124–133, 2000.

[9] Y. Hu and B. Panda. Mining Inter-transaction Data Dependencies for Database Intrusion Detection. In *Innovations and Advances in Computer Sciences and Engineering*, pages 67–72, 2010.

[10] W.L. Low, J. Lee, and P. Teoh. DIDAFIT: Detecting Intrusions in Databases Through Fingerprint Transactions. In *Proceedings of the 4th International Conference on Enterprise Information Systems*, pages 264–269, 2002.

[11] E. Bertino, A. Kamra, and J.P. Early. Profiling Database Application to Detect SQL Injection Attacks. In *IEEE International Performance Computing and Communications Conference (IPCCC)*, pages 449–458, 2007.

[12] R. Sekar, A. Gupta, and J. Frullo. Specification-based Anomaly Detection: A New Approach for Detecting Network Intrusions. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 265–274, 2002.

[13] R. Agrawal, T. Imieliiski, and A. Swami. Mining Association Rules Between Sets of Items in Large Databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, 1993.

[14] R. Agrawal and R. Srikant. Mining Sequential Patterns. In *Proceedings of the 11th International Conference on Data Engineering*, pages 3–14, 1995.

Mostafa Doroudian received the B.S. degree in computer engineering (software) from Mazandaran University of Science and Technology, Mazandaran, Iran, in 2010, and the M.S. degree in information technology engineering (information security) from Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran, in 2012. His research interests include areas of database security, intrusion detection, network security, computer forensics, and alert correlation systems.

Hamid Reza Shahriari is currently an assistant professor at the Department of Computer Engineering and Information Technology in Amirkabir University of Technology, Tehran, Iran. He received his Ph.D. in computer engineering from Sharif University of Technology in 2007. His research interests include information security especially in vulnerability analysis, security in e-commerce, trust and reputation models, and database security.

The ISC Int'l Journal of Information Security
ISeCure

July 2014, Volume 6, Number 2 (p. 196)
http://www.isecure-journal.org

Persian Abstract

یک رویکرد ترکیبی برای تشخیص نفوذ پایگاه‌داده در سطوح تراکنش و میان‌تراکنش

مصطفی دورودیان[1] و حمیدرضا شهریاری[1]

[1]دانشکده مهندسی کامپیوتر و فناوری اطلاعات، دانشگاه صنعتی امیرکبیر، تهران، ایران

امروزه اطلاعات نقش مهمی در سازمان‌ها ایفا می‌کند. اطلاعات حساس اغلب در پایگاه‌های داده ذخیره می‌شوند. مکانیزم‌های سنتی از قبیل رمزنگاری، کنترل دسترسی و تصدیق اصالت سطح بالایی از اطمینان را فراهم نمی‌کنند. بنابراین وجود سامانه‌های تشخیص نفوذ در پایگاه‌داده امری ضروری به‌نظر می‌رسد. در این مقاله یک سامانه تشخیص نفوذ پایگاه‌داده به‌منظور تشخیص حملات در سطوح تراکنش و میان‌تراکنش (وظیفه کاربر) ارائه می‌شود. برای این منظور ابتدا یک روش تشخیص در سطح تراکنش ارائه می‌شود که مبتنی بر توصیف تراکنش‌های مورد انتظار در سطح برنامه‌های کاربردی پایگاه‌داده می‌باشد. سپس در سطح میان-تراکنش یک روش تشخیص ناهنجاری پیشنهاد می‌شود و از رویکرد داده‌کاوی برای یافتن قوانین وابستگی و دنباله میان‌تراکنشی استفاده می‌شود. از مزایای این سامانه نسبت به سامانه‌های تشخیص نفوذ پایگاه‌داده پیشین، توانایی تشخیص رفتارهای مخرب در هر دو سطح تراکنش و میان‌تراکنش می‌باشد. همچنین این سامانه با بهره‌گیری از مزایای ترکیب رویکردهای تشخیص مبتنی بر توصیف و تشخیص مبتنی بر ناهنجاری موجب به حداقل رساندن هشدارهای مثبت کاذب و منفی کاذب می‌شود. آزمایش‌هایی به‌منظور ارزیابی درستی عملکرد سامانه پیشنهادی انجام شده است. نتایج ارزیابی‌های عملی حاکی از بالا بودن میزان درستی عملکرد و سودمندی سامانه پیشنهادی می‌باشد.

واژه‌های کلیدی: تشخیص نفوذ، امنیت پایگاه‌داده، ماشین وضعیت، وابستگی میان‌تراکنشی، دنباله میان‌تراکنشی.