An Intrusion Tolerant Transaction Management Model for Wireless Storage Area Networks

Yacine Djemaiel, Noureddine Boudriga, Soukeina Zouaidi
CN&S Research Lab., University of Carthage
Tunisia
yacine.djemail@supcom.rnu.tn, noure.boudriga2@gmail.com

ABSTRACT: *Storage area networks is among the solutions that have been deployed in most enterprise information systems through the world by ensuring the processing of corporate data taking benefit of the high speed and the separation of provided services from processed data. Due to the great need of such infrastructures, a wireless connectivity have been added to these networks in order to ensure more flexibility and mobility for interconnected components in order to deal with environment constraints in addition to advanced security strategies defined for critical services and data and that have lead to a new kind of networks called wireless storage area networks. Such emerging networks integrate in most cases transaction-based applications that are exposed to several security threats that aim to prevent the correct processing of transactions and engender damage to interconnected components. In this paper, we propose a novel intrusion tolerant transaction management model for wireless storage area networks. The proposed model defines the secure flex transaction concept and is based on intrusion and detection strategy and compensability feature to manage transactions in WSAN either if this environment is compromised. Moreover, this model uses the Predicate Transition Net as a tool to control the executability of secure flex transactions. In order to illustrate the behavior of the proposed model and their security capabilities, a case study is defined for a transaction-based e-payment service for an online commerce company integrated in a monitored wireless storage area network that is exposed to a set of security attacks.*

Keywords: Transaction, Flex, Compensation, Intrusion Tolerance, PTN, WSAN

Received: 26 May 2013, Revised 13 July 2013, Accepted 15 July 2013

1. Introduction

SANs (Storage Area Networks) have shown these last years a great use in enterprise networks since it provides a high performance network that connects available storage devices holding enterprise data and the back-end servers. As an evolution to the SANs, the Wireless Storage Area Networks, called WSANs have emerged taking benefit of the SANs and the flexibility of the wireless links associated to these networks. These networks are needed nowadays since they ensure the deployment of high speed links between available services and the required data that may be held in different area of the organization where making a wired link seems to be either difficult or impossible. In addition, handled storage devices in this case, may move from an area to an other according to the provided security level that is affected to the area throughout the day (in the case of a

scheme that affects security levels in a dynamic manner for the different defined activity periods), or when a security strategy is defined recommanding that the physical location of critical data should be dynamic, meaning that the location of storage devices holding such data follows a defined moving scheme throughout the different periods in order to prevent the intruder from locating the data needed to run its attack. Integrating such technologies in enterprise networks may expose them to a set of issues that are related to the processing of specific kind of requests such as transactions in addition to security issues. In this context, this paper proposes an intrusion tolerant transaction management model for such networks integrating transaction-based applications. The contribution of this work is three-fold: first, we propose a novel architecture for WSAN including a set of defined components that are needed to ensure the processing and the management of long running online transactions. Second, the secure flex transaction concept is introduced to be adopted as a transaction model in the proposed WSAN architecture that may be exposed to a set of security threats. Third, the paper introduces an intrusion tolerant transaction execution and management model for WSAN.

Attacks in transaction-based environments is among the topics that have been well investigated by research community. Intrusion tolerance is a feature proposed to cope with malicious behavior in many environments aiming to ensure the continuity of the processing of transactions and to achieve the commit state. In [3], an intrusion detection and tolerance system have been proposed for storage area networks. The proposed solution allows the execution of malicious transactions in a separate area called VCA (Virtual Contaminated Area) in the storage device to prevent the data written by legitimate transactions to interfere with data written by suspicious ones. This system is applied in a wired environment involving only traditional transactions and it is unable to deal with complex transaction model and to ensure the recovery after an occurring failure. In this context, a solution taking into account the particularities of a wireless environment was proposed in [4]. This scheme is based on the generation of a set of static and dynamic rules which takes into account the transaction specificities to detect malicious actions and uses partial rollback mechanism to recover malicious activity. This work considers only the traditional transaction model since the used model didn't highlight advanced features such as alternatives, dependencies and compensation.

The intrusion tolerance concept have been studied for database systems. In this context, an intrusion tolerant database system was proposed in [6]. This system uses the combination of two intrusion tolerance techniques which are attack isolation and multiphase damage confinement. The first technique ensures the isolation of suspicious activities in a separate disk until the proof of the innocence of the user. The second technique ensures that no damage will spread after detecting a malicious transaction by preventing the access to the damaged data. The proposed system has three distinct characteristics. The first is isolation time of the malicious process. The second is significance of data object by the use of weight factor associated to each data object. The third characteristic is using a waiting queue for each suspicious data object. This work didn't consider a transaction to be executed in distributed manner with dependencies between different subtransactions.

Moreover, the modelling of transactions is among the research fields that have been well investigated. In this context, [7] has proposed an advanced transaction model, called flex transaction. This model has added three extensions to the traditional transaction model which are alternatives, dependencies and compensation. Alternatives signifies that the same task of a given transaction can be ensured by one or more subtransactions which enables the processing of transactions in a flexible manner even if some components fail. Dependencies means that the execution of a subtransaction can depend on the failure or the success of another subtransaction. The compensation means that the effect of a subtransaction can be semantically undone when its processing fails or additional factors may arise and that prevent it to move to a commit state. In spite of the additional features added to the flex model, the execution algorithm of this transaction model didn't take into consideration malicious transactions.

The paper is organized as follows. Section 2 presents the WSAN model to be used as a framework for integrating transaction-based applications and that ensures the processing and the interaction between the different components that belong to such network. Section 3 provides a formal model for the proposed intrusion tolerant transaction model called secure flex transaction for WSANs. Section 4 introduces the strategies on which the transaction management is based on, which are intrusion detection and tolerance strategy and compensation strategy. The control execution of the secure flex model using a predicate transition net is detailed in Section 5. Section 6 provides a discussion of the different adopted mechanisms to ensure a secure transaction management in WSAN. The next section presents the intrusion tolerance capabilities of the proposed scheme through a case study associated to a defined WSAN integrating an e-payment service. Section 8 concludes the paper and provides some perspectives as a future work.

2. WSAN model

The proposed WSAN model is based on SAN system which was proposed in [1] consisting of storage devices, specialized networks for connecting storage devices to servers and a management layer for setting up and maintaining connections between these servers and data storage devices. The SAN architecture is composed of the following three basic components:

- **SVC (SAN Volume Controller):** it is the main component that manages the interactions between the server and SDs. It virtualizes storage devices into a common pool and allocates storage to host systems from that common pool.
- **SDs (Storage Devices):** which are devices used for storage such as disk array, tape libraries. They are accessible to the servers, so that, they appear like locally attached devices to the operating system.
- **Servers:** are connected to SAN which allow clients to access data that are located at the SDs.

We propose to enhance this model by using a SRB (Storage Resource Broker) instead of the SVC and attaching wireless access points to the SRB and the relay equipments (switch, bridge, hub, etc.) in order to make more flexibility in the placement of storage devices and servers and reducing the use of cabling.

The SRB that will be used in our WSAN architecture was proposed initially in [2]. This component provides seamless access to data stored on a variety of storage resources including file systems, database systems, and archival storage systems. The SRB includes an API for accessing data, a MCAT (Metadata CATalog) for organizing data collection attributes, one or more SRB master processes and SRB agent processes. Each SRB master controls a distinct set of PSRs (Physical Storage Resources). The MCAT is generally a database which manages mapping of data objects to storage resources. In this context, the architecture of the proposed WSAN is depicted by Figure 1.

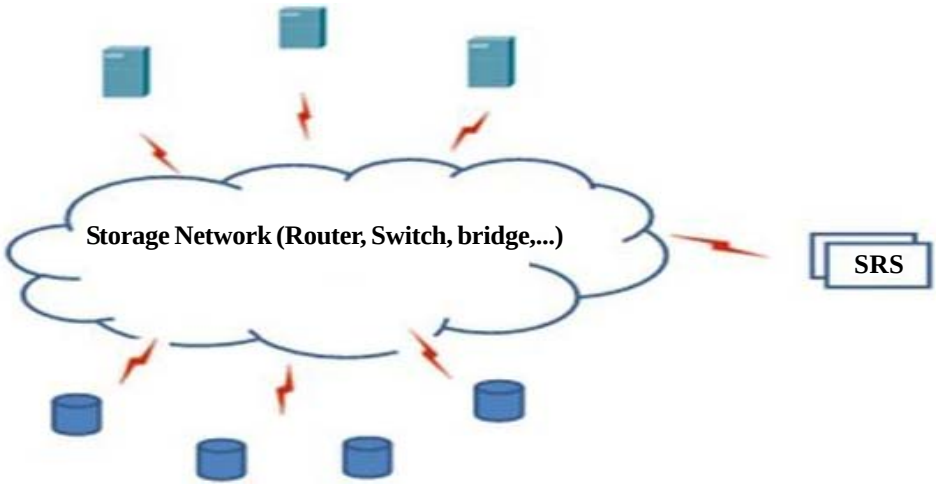

Figure 1. Wireless Storage Area Network

Three additional components are added to the SRB in order to support the particularities of such novel WSAN environment that integrates transaction-based applications. The first component is the **Transaction Splitter** that uses the information stored at the MCAT to split transactions into multiple subtransactions based on data location. The second component is the **PTN Manager** that ensures the control of the execution of transactions using PTN (Predicate Transition Net). The third component is **TST (Transaction Status Table)** which keeps the progress execution status of transactions. The SRB ensures the management of transactions in such environment by controlling the commit and the rollback operations that may be performed locally or globally according to the transaction anatomy. It also deals with the possible threats and particularities related to the wireless connectivity (e.g. the node is temporarily out of coverage). In order to enhance the processing in a WSAN environment, the SRB monitors the signal level in order to serve the requested data from the *SD* that is the most closet to it. Note that the WSAN integrates more than one SRB to make the system more reliable in case of unavailability of the current one due to an attack.

3. Secure flex transaction model

We have adopted the flex transaction model, that is proposed in [7] to take advantage from its alternatives and compensation properties in order to allow more execution alternatives of the transaction and to be able to undo the effect of malicious subtransactions. In this model, the transaction is viewed as a set of tasks and the user is allowed to specify for each task, a set of functionally equivalent subtransactions, each of which when completed will accomplish the task. The execution of a flex transaction succeeds if all its tasks are accomplished. The subtransactions of a flex transaction are executed in parallel. After the completion of a subtransaction, it is checked whether or not an acceptable execution state has been reached. If an acceptable state has been reached, the flex transaction is terminated with a committed status. When the flex transaction is terminated without reaching an acceptable state, then a feedback is required from the user to decide whether to commit or not with partial result. We propose to enhance the flex transaction model by introducing two types of subtransactions which are tolerated *T* and non tolerated *NT*. A subtransaction is said to be of type *T* if it is tolerable, it is of type *NT* , if it is not tolerable by matching one of the rules that are defined in Section 4.1. We assume that all the subtransactions of the secure flex are compensatable in order to make possible the compensation of each malicious effect.

3.1 Form of a Secure flex Transaction

A secure flex transaction *T* is formally defined as 4-tuple (B, D, Π, f) where:

$B = \{t_1, t_2, ..., t_n\}$ is a set of subtransactions called the domain of *T*.

$D = S \cup F$: set of partial order on *B*.

S is a partial order on *B* called the success order of *T*.

$t_i \prec t_j \in S$ means that the subtransaction t_i has to be successfully executed before the subtransaction t_j may be started. *F* is a partial order on *B* called the failure order of *T*.

$t_i \prec t_j \in F$ means that the subtransaction t_j has to be executed only after the subtransaction t_i is executed and failed.

Π is a set of external predicates on *B*.

f is an n-ary boolean function defined on the set {1, 0} and is called the acceptability function of *T*.

$$f(X) = \begin{cases} 1 \text{ if an acceptable executionwas reached} \\ \quad \text{with legitimate or tolerated subtransactions} \\ 0 \text{ if no acceptable execution state is reached} \end{cases}$$

According to this proposed model, the processing of the subtransactions of a given transaction may be handled even if there are some SDs that are involved in the processing and are not available due to for example, damage or loss of coverage using the defined failure order.

The selection of adequate SDs by the SRB for the execution of subtransactions is modelled through a set of external predicates added to Π. Two external predicates was introduced in the secure flex transaction model which are:

trust_SD(t_i): this predicate function is used to evaluate if the required SD for the execution of t_i is trust or no.

available_SD(t_i): this predicate function is used to evaluate if the required SD for the execution of t_i is available in term of coverage or no.

3.2 Execution states

The transaction execution state *X* of a transaction *T* composed of *n* subtransactions is an m-tuple ($x_1, x_2, ..., x_n$), where:

$$x_i = \begin{cases} N \text{ if subtransaction } t_i \text{ has not been submitted for execution}: \\ E_L \text{ if } t_i \text{ is currently being executed and that is considered as legitimate} \\ E_T \text{ if } t_i \text{ has been identified as suspicious and tolerated and currently being executed at V CA} \\ S_T \text{ if } t_i \text{ is tolerated and successfully completed at V CA} \\ S \text{ if } t_i \text{ has successfully completed at V HA} \\ F \text{ if } t_i \text{ has failed or completedwithout acheiving its objective} \end{cases}$$

Three states are added compared to the initial proposed flex model in order to ensure the monitoring and the intrusion tolerance of identified suspicious subtransactions. The first state, denoted as E_L is associated to the running subtransactions that are not identified as suspicious by the deployed security solutions at the SRB which are detailed in 4.1. The state E_T is related to the running subtransactions that are identified as suspicious and that are processed during the intrusion tolerance process. The last state denoted S_T is associated to the running tolerated subtransactions that have been completed. Their effects are applied on the VCA and not yet applied in the VHA (Virtual Healthy Area). These two areas are more detailed in section 4.1.

The commit of the whole transaction requires that all the subtransactions have yet achieved the state *S*.

3.3 Execution rules of secure flex transaction

The processing of a secure flex transaction takes into consideration the execution dependency relationships that may characterize its subtransactions based on their execution states values, in addition to the evaluation of their corresponding predicates. To ensure this goal, a secure flex transaction handled in WSAN is modeled as follows:

Let $T = (B, D, \Pi, f)$ be a secure flex transaction and t_i be an element in *B*.

The set *Pdept* (t_i) is the subset of *B* constituted by all elements *t* of *B* such that t_i is positive dependent on *t*.

The set *Ndept* (t_i) is the subset of *B* composed of all elements *t* of *B* such that t_i is negative dependent on *t*.

Let $X = (x_1,..., x_n)$ be an execution state of transaction *T*.

The subtransaction t_i is executable at state *X* if the following four assertions are satisfied:

1. $x_i = N$;
2. *For all j such that* $t_i \in$ *Pdept* (t_i), $x_j = S$;
3. *For all j such that* $t_j \in$ *Ndept* (t_i), $x_j = F$;
4. *For each external predicate P, P* (t_i) *is true*

4. Secure flex transaction management in WSAN

An intrusion detection and tolerance strategy is required in order to ensure a secure flex transaction management in WSAN. In this section, we detail this strategy in addition to the compensation process to deal with the failure cases related to transactions processing and malicious behavior.

4.1 Intrusion detection and tolerance strategy

Intrusion detection mechanisms are used in the WSAN in one hand at the level of SRB to discover attempts to gain unauthorized access to the stored data in the SDs, in other hand at the level of SDs to detect corrupted data.

At the SRB, the intrusion detection and tolerance process is based on two basic routines which are: *check_security* and *check_tolerance*. The *check_security*() function checks if a subtransaction is malicious or legitimate before submitting it to the adequate *SD*. This detection is based on a set static and dynamic rules that represent the possible violations that may be happen against the set of *SD*s (e.g. The attempt to alter data or access to non authorized data stored in an *SD*), the *SRB* (e.g. The intruder

tries to modify the behavior of the SRB using a malicious transaction), the wireless connectivity (e.g. the out of coverage for a needed SD) and the subtransactions processing (e.g. a denial of service attack against the *SD*s attached to the WSAN). These rules represent the profile associated to a malicious activity against the WSAN components. The *check_tolerance* routine checks if a suspicious subtransaction can be tolerated to be executed at the SD. The intrusion tolerance process is not initiated if one or several of the following constraints on each resource *R*, handled by the suspicious subtransaction, are true:

- *R* is handled by a real time critical service.
- *R* holds confidential data related to running service (e.g. service configuration file, password file, etc.).
- A concurrent access to *R* is performed by several malicious subtransactions (only the first subtransaction is authorized).
- The requested *R* is corrupted and does not been yet replaced by the correct copy.

Additional rules are added to the aforementioned rules in a dynamic manner when a monitored malicious activity tries to move the WSAN to a failure state. These intrusion tolerance rules are also generated based on the dependency relationships that exist between the different running transactions and their related sub transactions.

At the SDs, an intrusion tolerance strategy is used which was defined initially in [5]. This strategy aims to divide the SD disk in two areas: the first area, called VHA, where legitimate subtransactions are executed. The second area is the VCA where the suspicious subtransactions are executed. The subtransaction is checked against intrusion detection rules before transferring its effect to the VHA. Once its legitimacy is verified, its effect is retranscribed at the VHA. The separation between these two areas is simply based on the upholding of a virtual separation file which maps every SD disk block to healthy or contaminated area. The aforementioned file is stored in the protected area inside the SD to be accessed in a highly secure manner. This area is only accessible by the administrator of the WSAN through the SRB after accomplishing a strong authentication check. For each performed action on the SD resource *R*, a set of information are hold and that enables the SRB to move the modifications on resources to the VHA or to undo the change by enabling the compensation process. Information attached to the handled resources are the following: *id_R* (*id_transaction*, *id_subtransaction*, *operation*, id_R_i)∗. ∗ means that these attributes may be repeated as many as *R* is processed by different subtransactions and by considering the order of their execution. *i* represents the order of the different instances created for *R* where the requested operation introduces modification on it. For example, if a resource R_1 is processed by three dependent transactions that are respectively: T_1, T_2 and T_3 where the set of involved subtransactions (t_{12}, t_{24} and t_{32}) handle R_1 respectively by performing the following operations: r*ead*, *write* and *read*. An entry associated to this dependency scheme is added including the following fields: id_R_1 (id_T_1, id_t_{12}, *read*, $id\_R_{1_1}$) (id_T_2, id_t_{24}, *write*, $id\_R_{1_2}$) (id_T_3, id_t_{32}, *read*, $id\_R_{1_3}$).

4.2 Compensation strategy

The execution of a secure flex transaction uses compensation mechanism according to two ways as detailed below:

4.2.1 Online compensation

This mechanism is used when the intrusion tolerance process is terminated without achieving the success state *S* for a suspicious subtransaction that has been executed at the VCA but detected as malicious. In this case, all modifications made on handled data items by this subtransaction are cancelled using the information attached to each handled data detailed in section 4.1 to restore the correct copies.

4.2.2 Offline compensation

This mechanism is used to undo the effect of the committed subtransactions for two possible cases. The first case is associated to subtransactions which their parent transaction has not yet reach the final commit status i.e. when the termination condition is satisfied during the execution of the transaction and no acceptable execution state is reached and the client rejects partial results. In this case, the compensation is required to undo the effect of the committed subtransactions. Therefore, the SRB issues the corresponding compensating subtransactions to the appropriate SDs.

In the second case, this mechanism is used to undo the effect of the committed subtransactions that have accessed a corrupted data. When a subtransaction appears malicious at the termination of execution at the VHA, therefore the committed subtransactions that have accessed to the data items updated by it must be compensated. In this case, a dependency graph is used to determine the dependency between the subtransactions that they read or write the same data items. This graph is constructed based on

the definition given in [8] which states that a subtransaction t_{ik} is dependent upon subtransaction t_{jl} in a history H if there exists a data item x such that:

1. t_{jl} reads x after t_{ik} has updated x; and

2. There are no subtransactions that update x between the time t_{ik} updates x and t_{jl} reads x.

This graph is constructed based on the information associated to the instances of handled resources detailed in Section 4.1.

Figure 2 shows an example of dependency graph. It means that the subtransaction t_{21} of the transaction T_2 depends on the subtransaction t_{11} of the transaction T_1, the subtransaction t_{33} of the transaction T_3 depends on the subtransaction t_{11} and the subtransaction t_{32} of T_3 depends on the subtransaction t_{21} of the transaction T_2. For example, when the execution of t_{32} is terminated, it could not be committed until ensuring that its preceding subtransaction in the graph t_{21} is innocent. If t_{21} turns out malicious then t_{21} and t_{32} must be compensated.

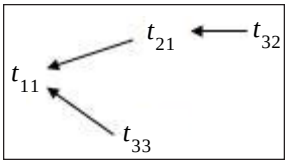

Figure 2. Dependency Graph

5. Checking the executability of secure flex transaction by predicate transition net

A PTN (Predicate Transition Net) is used as a tool to control the execution of secure flex transactions. In this section, we start by modelling the secure transaction using the PTN then we detail the execution of the secure flex transaction according to the proposed approach.

5.1 Modeling the secure flex transaction model using PTN

A predicate transition Net consists of: 1) H: set of places, 2) K: set of transitions, 3) L: set of arcs, 4) $\mathcal{K}$: mapping of the set of transitions into the set of formulae of the first order logic, and 5) M_0: mapping of places: it assigns to each place h in H a set of symbolic characters (called tokens).

Let $T = (B, D, \Pi, f)$ be a given secure flex transaction, the associated predicate transition nets $PTN(T) = (H, K, L, \mathcal{K}, M_0)$ is defined as follows:

If $B = (t_1,...,t_n)$ then $H = \{a_1,...,a_n, b_1,.., b_n\}$ where we assume that for all i and j, the symbol a_i is equal to the symbol b_j iff t_i is positively dependent on t_j.

$K = B$: for each subtransaction t_i, a transition tr_i is associated.

$L = \{(a_i, tr_i) \mid i \leq n\} \cup \{(tr_i, b_i) \mid i \leq n\}$

$\mathcal{K}$: $\mathcal{K}(t_i) = \left(\bigwedge_{t \in Ndept(t_i)} F(t, t_i) \right) \wedge \left(\bigwedge_{q \in \Pi} q(t_i) \right)$ where $F(t, t_i)$ is the negative dependence predicates for any t in $Ndept(t_i)$ and $q(t_i)$ is the external predicates for any q in Π

Note that the external predicates introduced in the secure flex transaction model: $trust\_SD(t_i)$ and $available\_SD(t_i)$ described in 3.1 are the basic predicates that must be validated before the submission of a subtransaction t_i to the SD.

5.2 Execution of secure Flex transaction

During the execution of secure flex transaction, two major functions are invoked to ensure both the intrusion detection and tolerance related to running transactions in a WSAN. The *check_security*() function returns a decision related to the running sub-transaction if it is malicious or legitimate. This detection is based on a set static and dynamic rules that represent the

possible violations that may be happen against the set of *SD*s (e.g. The attempt to gain alter data or access to non authorized data stored in an *SD*), the *SRB* (e.g. The intruder tries to modify the behavior of the *SRB* using a malicious transaction), the wireless connectivity (e.g. the out of coverage for a needed node) and the transaction processing (e.g. force a failure for a set of dependent transactions based on a denial of service attack against either the servers or the *SD*s attached to the WSAN). These rules represent the profile associated to a malicious activity against the WSAN components. In order to ensure the intrusion tolerance process, the *check_tolerance* () function provides the decision if the transaction or the sub transaction may be tolerated or not. This decision is achieved based on the check of a set of intrusion rules that consider the dependency between the running transactions in addition to the set of requested operations during the execution of a transaction schedule. Moreover, these rules consider the security level associated to the requested resources on *SD*s in addition to the performed operations. An example of an intrusion tolerance rule may be given and that may be related to a running subtransaction that includes a write operation on a file that is accessed by an intruder after issuing a read operation. In this case, the intrusion tolerance rule allows a write by a legitimate subtransaction, while the read operation performed by the next sub transaction that is identified as malicious, will be performed against the copy of the file before applying the modifications made by the write operation. This copy of the file is located at the VCA. The access to a file may be rejected (end of the intrusion tolerance process) if the file have a high security level such as a service configuration file or a password file. The intrusion detection and tolerance rules are stored in an SD where the access is grant only for the SRB through the processing of the aforementioned security functions.

The execution of secure flex transaction is resumed in the following steps:

Step 1: For each new incoming transaction *T* at the SRB, the *Transaction Splitter* queries the *MCAT* about the location of data items on which its operations will be executed. Based on this information, the transaction is decomposed into multiple subtransactions $\{t_1,...,t_n\}$, where each one to be executed at a particular storage device. Then, the different acceptable execution states are generated.

Step 2: The PTN manager determines the set of enabled transitions i.e. the set of transitions which their input places contain tokens, then it deduces the set of executable subtransactions i.e. the correspondant set of transitions which their predicates are valid.

Step 3: Before submitting each subtransaction belonging to the set of executable subtransactions, it is checked by the use of the function, *check_security* (). If it seems to be malicious, then it will be checked if the requested resources match the tolerance rules or no using the function *check_tolerance*(). If it can be tolerated, then it can be submitted to the adequate SD else no.

Step 4: Upon reception of a subtransaction at the SD, if it is legitimate, then it is executed at the VHA, else if it is tolerated, then it will be executed first at the VCA. Then, if the output of its execution seems to be correct, then the subtransactions output are retranscribed at the VHA.

The execution results of a subtransaction at the VHA are stored in a way such that it can be later either be durably stored or compensated.

Once the execution of a tolerated subtransaction terminates at the VCA without causing damage, the SRB is notified and the execution state of the subtransaction moves to S_T. When its execution is finalized at the VHA, the execution state changes to *S*.

In the same way, when a legitimate subtransaction is completed and committed successfully at the VHA, the SRB is notified and its execution state becomes *S*.

Note that, the execution of the transaction terminates when any of the following conditions occur: 1) An acceptable execution state is reached, 2) None of the subtransactions is executable and no subtransaction is currently executing, and 3) The timeout is reached.

6. Security Analysis

In this section, we discuss the different mechanisms used to make the transaction management secure in the WSAN.

A secure flex transaction may be exposed to a set of attacks that may threat the data stored at WSAN SDs. For example, an

intruder can be able to inject malicious instructions in processed transactions or try to prevent the normal processing for either the SRB or the attached SDs.

Therefore, to reinforce security at the whole WSAN, intrusion detections mechanisms are used at both the SRB and SDs. At the level of SRB, the basic execution algorithm of the flex transaction was enhanced by adding a *check_security* () function that determines if a subtransaction is legitimate or not before submitting it to the SD and a *check_tolerance* () function that checks if a suspicious subtransaction can be tolerated or no by verifying if the re-quested resources match the intrusion tolerance rules. The efficiency of the proposed scheme is dependent on the intrusion detection capabilities that provide the way to identify for which resources and subtransactions, the intrusion tolerance process should be initiated. In this context, the *check_security* () and *check_tolerance* () routines should be compromise independent in order to ensure their correct processing even if the SRB is being compromised.

At the SD, an IDS is used to assess the output of the suspicious subtransactions at the VCA and the output of the legitimate subtransactions at the VHA. So, even if a legitimate subtransaction is attacked during its transmission from the SRB to the SD, the IDS can detect the intrusion, therefore the SD could not be damaged.

In order to make the proposed transaction processing model more reliable in case of unavailability of its main compo-nent SRB (e.g DoS attack), a process of election of a new SRB is triggered in order to ensure the continuity of service for the overall WSAN by initiating the migration of used structures such that the MCAT and TST to the new selected SRB and informing the servers and SDs.

Additional predicate functions were introduced in the secure flex model to make the processing of subtransactions more reliable. *trust_SD* () is used to check if the required SD is trust or not and *available_SD* () is used to check the availability of the required SD in terms of coverage before submitting the subtransaction.

7. Case study

We consider a WSAN that ensures electronic payment service for an online commerce company. This company offers to their loyal customers multi-store gift cards. In this WSAN, a customer could buy multiple items from multiple providers, his purchase order is directed to the SRB through a server. The SRB acts as an intermediary to help the customer to make purchase and payment with different providers simultaneously as a single payment transaction. The set of servers connected to the WSAN are available for customer in an airport for the different landside areas (parking, terminals, etc.) and from the aircraft when it is in the airside areas. The customer may be served from different servers during his stay at the airport.

Let consider a customer which issues a multi purchase order to buy two items I_1 and I_2 using a multi-store gift card. He specifies that he wants to buy I_1 with preference from the provider P_1 then P_1' and he boughts I_2 only if I_1 is available. The SRB selects the associated SDs that are needed to serve such transactions by selecting from the list of SDs that provide the same data (if a redundancy mechanism is deployed), the nearest one to the SRB by considering the different characteristics associated to the wireless link such as, the signal level.

We propose the following secure flex transaction to formalize the customer transaction denoted *T*.

$B = \{t_0, t_1, t_0', t_2\}$ where t_0 checks the validity of the gift card and its balance; t_1 purchases I_1 from P_1; t_1' purchases I_1 from P_1 if t_1 fails; t_2 purchases I_2 from P_2 if t_1 or t_1' succeeds.

$S = \{t_0 \prec t_1, t_0 \prec t_1, t_1' \prec t_2, t_1' \prec t_2\}$

$F = \{t_1 \prec t_1'\}$

$f(x_1, x_2, x_3, x_4, x_5) = (x_0 \wedge x_1 \wedge x_2) \vee (x_0 \wedge x_1' \wedge x_2)$

The corresponding PTN associated to this transaction is depicted by Figure 3.

Upon the reception of *T* at the SRB, the transaction splitter queries the MCAT about the location of the requested data that may be available in different SDs connected to the WSAN, then it splits it into multiple subtransactions based on data location. After

that, its execution starts using the proposed algorithm based on PTN.

At the beginning, the enabled set of transitions contains only the transition corresponding to the subtransaction t_0. Since the predicate of this transition is valid, this subtransaction can be executed. t_0 is submitted then to the gift card SD to check if the used gift card for payment is valid or no and if its balance is enough or no. The gift card is valid with a sufficient balance, therefore t_0 is successfully executed at the SD. Upon reception of the execution result of this subtransaction at the SRB, it updates its execution variable state to *S* and fires its corresponding transition. Therefore, *n* othe enabled set is $\{t_1, t_1'\}$. Since t_1' is negative dependent on t_1, the SRB tries to execute first t_1. After checking that its execution predicates are valid and that it is legitimate by running the *check_security* () routine, it is submitted to the Provider P_1 SD. Assume that the execution of t_1 fails, therefore its execution variable state changes to *F*. The SRB initiates the scheduling of t_1'. First, it checks if its execution predicates are valid. Assume that they are, it then runs the *check_security* () routine that identifies it as legitimate. After that, the subtransaction is submitted to the Provider P_1' SD and the execution of t_1' succeeds. Since t_1' is successfully executed, its execution variable state is updated to *S* and its corresponding transition is fired. The enabled set contains only the transition corresponding to t_2 and the corresponding predicate of t_2 are evaluated to true. After checking its legitimacy (decision provided by the *check_security* () routine), the SRB forwards it to the Provider P_2 SD. The execution of t_2 succeeds. The execution variable state of t_2 is updated to *S* and the corresponding transition is fired. An acceptable execution state (t_1', t_2) is reached in this case. The result is forwarded to the customer.

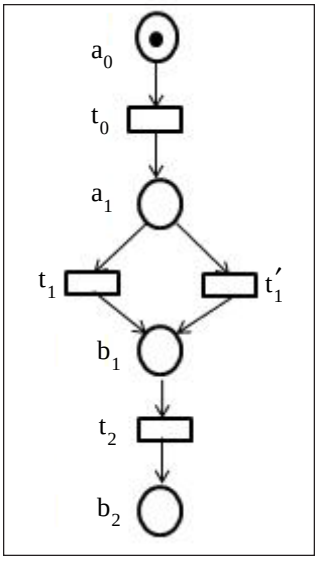

Figure 3. Predicate Transition Net for T_1

We consider now the same case described above with a malicious customer. If the cost of items are not encrypted before they are sent to the customer, the latter can modify the price of items. Assume that the customer succeeds to modify the transaction details (e.g. the price of item I_2). So, when the SRB runs the *check_security* () routine for the subtransaction t_2, it is detected as malicious and could not be tolerated because it makes lost for the provider. Else, if for example the SRB tolerates the execution of t_2, it will be executed first in VCA, the IDS in this area detects the malicious behavior therefore, its effect is undone (by the online compensation process). In this case, when the subtransaction t_2 could not be committed, the SRB requires a feedback from the customer to decide whether to accept partial results or no in order to make decision about the global commit or abort of *T*. If the customer refuses, the offline compensation process is triggered to undo the committed subtransaction t_1'.

During the processing of transactions, the SRB could be the target of a denial of service attack. In this context, the process of selection of a new SRB among the available ones is triggered. Once the selection is done, the required structures MCAT and TST are migrated to the new SRB, then the remaining SDs and servers connected to the WSAN are notified.

8. Conclusion

In this paper, we have proposed an intrusion tolerant transaction execution scheme for WSAN. The proposed scheme is an enhancement to a previous defined flex model. This scheme considers the particularities of the WSAN in addition to the set of constraints related to transaction-based applications. In order to ensure the processing of transactions either if the WSAN is compromised, an intrusion tolerance process is defined that may be applied to such environments in addition to a compensation strategy that ensures the undo of either a malicious transaction that can not be tolerated or due to an introduced damage to running components. The behavior of the proposed scheme has been illustrated through a case study that ensures an e-payment service for an online commerce company ensured by a set of components attached to a WSAN. The proposed scheme may be enhanced by supporting additional WSAN threats in addition to the extension of the proposed scheme in order to support interconnected WSAN environments.

9. Appendix

Algorithm 1 provides the execution control scheme for secure flex management model that is an enhancement to the provided algorithm detailed in [7].

Algorithm 1 The execution control algorithm for the secure flex model

```
procedure evaluate_ PTN (PTN, f, t0, R)
Initialize timeout mechanism with timeout interval t0;
begin
    x ← (N, N, N,..., N)
  on timeout flex _abort;
  ε ← φ / * Enabled set */
  ν ← φ / * Executable set */
  G ← φ / * Scheduled set */
  Q ← empty;
  compute _enabled_set ε from PTN;
  compute executable_set U from the new enabled set ε
repeat
  G+ ← U − G;
  for each transition tri ∈ G+ do
    begin
        if (check_security (tri) = legitimate) then
         begin
            submit subtransaction ti to the SD;
            xi ← EL;
            G ← G ∪ {tri}
          end
         else /*check_security (tri) ≠ legitimate */
              if (check tolerance (tri) = true)
                begin
                    submit subtransaction ti to the SD;
                  xi ← ET;
                  G ← G ∪ {tri};
                end
    end
   on receiving response enqueue response in Q;
     while (Q = empty) do
      begin
          if (U = φ ) then begin
              flex_abort;
              exit;
          end
   end
```

References

[1] Banikazemi, M., Poff, D., Abali, B. (2005). Storage-based intrusion detection for storage area networks. *In*: 22nd IEEE/13th NASA Goddard Conference on Mass Storage Systems and Technologies, p. 118–127, Washing-ton,DC,USA. *IEEE Computer Society*.

[2] Baru, C., Moore, R., Rajasekar, A., Wan, M. (1998). The sdsc storage ressource broker. *In*: Proceedings of the conference of the Centre for Advanced Studies on Collaborative research (CASCON '98).

[3] Djemaiel, Y., Boudriga, N. (2008). Dynamic detection and tolerance of attacks in storage area networks. *In*: the proceedings of the IEEE AINA-2008 workshop, p. 377–382, Okinawa, Japan, March.

[4] Djemaiel, Y., Boudriga, N. (2009). Intrusion detection and tolerance for transaction based applications in wireless en-vironments. *In*: the proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS 2009), p. 1–8, May.

[5] Djemaiel, Y., Rekhis, S., Boudriga, N. (2008). Intrusion detection and tolerance: A global scheme. *International Jour-nal of Communication Systems*, 21 (2) 211–230.

[6] Falahiazar, Z., Rohani, M., Falahiazar, L., Teshnelab, M. (2012). Optimizing an intrusion tolerant database system using neural network. *Database Theory and Application*, 5, 83–98.

[7] Leu, Y., Elmagarmid, A., Boudriga, N. (1990). Specification and execution of transactions for advanced database applications. *Technical report*, Purdue University.

[8] LIU, P., AMMANN, P., JAJODIA, S. (2000). Rewriting histories: Recovering from malicious transactions. *Distributed and Parallel Databases*, 8, 7–40.