Mitigating insider threat in cloud relational databases

Qussai Yaseen1 *, Qutaibah Althebyan2, Brajendra Panda3 and Yaser Jararweh4

1 Computer Information Systems Department, Jordan University of Science and Technology, Irbid, Jordan
2 Software Engineering Department, Jordan University of Science and Technology, Irbid, Jordan
3 Department of Computer Science and Computer Engineering, University of Arkansas, Fayetteville, AR 72701 U.S.A.

ABSTRACT

Cloud security has become one of the emergent issues because of the immense growth of cloud services. A major concern in cloud security is the insider threat because of the harm that it poses. Therefore, defending cloud systems against insider attacks has become a key demand. This work deals with insider threat in cloud relational database systems. It reveals the flaws in cloud computing that insiders may use to launch attacks and discusses how load balancing across availability zones may increase insider threat. To mitigate this kind of threat, the paper proposes four models, which are peer-to-peer model, centralized model, Mobile-Knowledgebases model, and Guided Mobile-Knowledgebases model, and it discusses their advantages as well as their limitations. Moreover, the paper provides experiments and analysis that compare among the proposed models, demonstrate their effectiveness, and show the conditions under which they work with highest performance. Copyright © 2016 John Wiley & Sons, Ltd.

KEYWORDS
cloud computing; databases; insider threat; security

*Correspondence
Qussai Yaseen, Computer Information Systems Department, Jordan University of Science and Technology, Irbid, Jordan.
E-mail: qmyaseen@just.edu.jo

1. INTRODUCTION

Cloud computing refers to the use of the internet to host computer resources and applications instead of keeping them on local computers. It delivers services (applications) over the internet, and hardware and systems in data centers [1]. Resources on the cloud are fully managed by cloud providers and sold (leased) on demand. The management of such resources includes monitoring, provisioning, de-provisioning, workload balancing, and changing requests [2]. The services provided by the cloud include Infrastructure-as-a-Service, Platform-as-a-Service, and Software-as-a-Service (SaaS). Moreover, cloud relational database systems are offered by some providers such as Amazon [3], which enable insiders to create and use relational databases.

Using virtual machines to run applications is one of the main features of using the cloud, where cloud platforms host many applications (tenants). Adopting multi-tenancy reduces the operating cost by allowing powerful resources sharing among tenants. Managing virtual machines is required in order to achieve effective resources utilization. Load balancing is performed using live migration [4], where virtual machines are migrated from overloaded nodes to idle (or low-loaded) nodes. However, live migration may pose a delay in delivering services because it limits the availability during the migration process. Developing methodologies for efficient and low cost live migration has obtained a significant attention by researchers. A number of methods have been proposed for effective live migration such as Albatross [4] and Zephyr [5].

When talking about cloud computing, security raises up as one major issue and concern. Proving the security of data in the cloud is critical in gaining users’ trust on cloud providers. Multi-tenancy could be a vulnerability source. For instance, an insider may use shared resources to breach the security of other insiders’ tasks [6]. Moreover, the guarantee of protecting data stored in the cloud from the threat of cloud providers’ insiders is a main requirement by customers. Encryption is considered as one of the means that are proposed to protect data. Namely, the encryption methods CryptDB, Homomorphic Encryption, and Encryption Deterministic can execute the operations of relational databases queries on encrypted data [7]. These methods prevent the cloud providers’ employees from exposing users’ data even when customers’ queries are
executed. Besides protecting data, authentication of users is another major concern when moving to the cloud. Thus, the development of digital identity management systems is crucial for cloud computing [8]. The agreements between cloud customers and cloud providers regarding the security and offered services are set using service level agreements, which should be maintained by cloud providers.

Insider threat is one of the major problems that worry both organizations and individuals about cloud computing. Storing data in the cloud maximizes the number of insiders, which in turn, may increase insider threats. Moreover, securing data in the cloud against organizations’ insiders may require new mechanisms different than those used to protect data stored locally. According to different surveys, such as the Computer Security Institute (CSI) survey [9], Forrester Research [10], and the Information Security Breaches Survey (ISBS) survey [11], insider threat causes huge harm to individuals and organizations. The CSI survey [9] stated that the cost of data records lost to insider attacks is greater than those lost to outsiders. This is because that insiders are familiar with the system and attack the valuable records, while outsiders steal what they can access. In fact, little research has been performed on insider threat. Most of insider threat existing research focuses on the system level. Very little research has been conducted on insider threat at relational databases level. Knowledgebase is a serious source for insider threat. The knowledgebase (discussed in Section 3) of an insider represents the data items that the insider accessed and read their values. Insiders can combine the data items that they accessed (in their knowledgebases) with other data items that they can request to infer sensitive information. Researchers in [12–16] have discussed this problem and introduced methods to mitigate it. However, cloud databases have new flaws that may help insiders to bypass existing security methods and pose threat using their knowledgebases. One of these possible vulnerabilities is the migration (live migration) of insiders’ requests across availability zones and data centers because of load balancing. The live migration of insiders’ tasks may move those requests from places where insiders’ knowledgebases are stored in new nodes that do not have the knowledgebases. Thus, systems on the new nodes are no longer able to check knowledgebases to detect and prevent insider threat. This paper addresses this problem and suggests different models to mitigate it. To the best of our knowledge, this is a premier work that tackles this problem at cloud relational database systems (RDBMS). We should mention here that this paper is an extended version of the paper published in [17]. The contribution of this paper is as follows.

1. It demonstrates how knowledgebases-based existing insider threat preventing mechanisms can be bypassed by insiders in cloud RDBMS.
2. It proposes four models that can be used to mitigate the threat of knowledgebases in cloud RDBMS. Moreover, it discusses the environment factors that affect the performance of the models. In addition, it discusses how to manage the effect of updating data items in knowledgebases using the aforementioned proposed models.
3. The paper used the CloudExp Simulator [18], which is an extended version of the CloudSim Simulator, to test the proposed models and provides results and analysis that compare among the proposed models. The experiments show the performance of models under different environments parameters.

The rest of the paper is organized as follows. The next section introduces some related work. Section 3 discusses several issues about insiders and the threat that they might pose to the system. It also gives brief description about knowledgebases and their threat. Section 4 introduces the four proposed models to solve the problem. Other issues related to insider threat in cloud RDBMS are discussed in Section 5. Section 6 provides the experiments and discusses the results. Finally, Section 7 concludes the work.

2. BACKGROUND AND RELATED WORK

Cloud computing is a promising technology that provides large-scale and on-demand computing infrastructure. The spending of the US government on cloud computing projects will pass $15bn by 2015 [19]. Achieving low cost live migration is one of the goals of the research on cloud computing. Das et al. [4] introduced Albatross, an end-to-end technique for live migration in shared storage databases. Albatross maximizes the availability during a migration process by migrating the cache and the state of active transactions instead of stopping transactions at source nodes and restarting them at destinations nodes. Zephyr [5] minimized service interruption and increased the availability during live migration by using a synchronized dual mode. The proposed dual mode enables both the source and destination nodes to execute transactions simultaneously while the migration process is running. Zephyr transferred the tenant’s (migrated application) metadata to the destination to start executing new transactions; meanwhile, the source node continues executing the transactions that were active before starting the migration process.

Assuring cloud security is a key demand to guarantee customers’ trust because there are many security concerns arising with cloud computing. Researchers in [6,20] introduced a survey about the common security concerns that threaten cloud computing. Multi-tenancy is one of the problems discussed, where the data of many users reside at the same location. This technique may enable the intrusion into a user’s data by other users, which can be performed by exploiting applications bugs or injecting masked code into the Software-as-a-Service system. Takabi et al. [21] introduced a security framework for cloud computing environment. They tried to build a comprehensive model that preserves cloud security using identity management models, access control models, semantic heterogeneity among
policies from different clouds, and others. The intrusion problem in a cloud environment was also discussed by Arshad et al. [22] who proposed a methodology for intrusion detection based on clustering. Wang et al. [23] investigated the problem of data security in cloud data storage. They utilized the homomorphic token with distributed verification of erasure coded data to achieve storage correctness insurance. Hwang et al. [24] addressed outlines for an integrated architecture to guarantee security and privacy in cloud applications. Chow et al. [25] suggested extending control measures in the cloud by using trusted computing and applied cryptography.

Research in cloud databases is still in its early stages. Little research has been conducted in this field. Hacigumus et al. [26] introduced CloudDB, a data management platform in the cloud. CloudDB has several features that satisfy cloud environments. It maintains three types of data stores, which are row store, key-value store, and analytics store to satisfy different workload types. For instance, the analytics store is a read-optimized and a throughput oriented to efficiently handle OLAP workloads, while the key-store is used to achieve higher levels of scalability for read/write intensive workloads. Moreover, CloudDB uses both partitioning and replication techniques to achieve availability and scalability. Curino et al. [7] introduced a new comprehensive framework for relational database on the cloud, called Relational Cloud. It supports new models for efficient multi-tenancy to minimize the resources needed for a workload, an elastic scale-out model to satisfy growing workloads, and models to preserve database privacy. Furthermore, Relational Cloud involves techniques for efficient partitioning, replication, and migration to achieve maximum availability and reliability. Unlike other multi tenant databases, Relational Cloud does not mix the data of different tenants into a shared database or table. Instead, databases belonging to different tenants are run on the same database server. Hence, the cloud relational database service has been introduced by some cloud providers such as Amazon (Amazon Relational Database Service Amazon RDS [Seattle, WA, USA]) [3] and Microsoft (Microsoft SQL Azure [Redmond, WA, USA]) [27].

Most research that has been performed on insider threat focused on system level such as the work by Spitzner [28] and Althebyan and Panda [29]. Fuchs and Pernul [30] proposed a comprehensive system of securing identity management in order to minimize insider threat. Their proposed approach gathers cross-domain identity information, removes unnecessary accounts, and filters account data and access rights in order to prevent insiders from obtaining unauthorized information. Claycomb et al. [31] proposed a framework for directory virtualization in order to detect insider attacks against directory services.

Very little research has dealt with insider threat in relational databases. Yaseen and Panda [12–14] showed how insiders use their knowledges and dependencies among data items to pose threat. Furthermore, they proposed models to detect and prevent insider threat in relational database systems. Yaseen and Panda [15] discussed the importance of organizing access privileges when executing transactions in mitigating insider threat. They addressed the risky role of knowledges that insiders can exploit to cause harm to systems. To the best of our knowledge, this paper is a premier research in the field of insider threat in cloud relational databases.

3. INSIDER THREAT

The knowledges of insiders and dependencies among data items, play a major role in insider threat in relational databases. The following section discusses how insiders can use dependencies and knowledges to launch attacks.

3.1. The threat of knowledges

The knowledges of an insider consists of data item values that she/he has previously accessed. That is, it represents the history of accesses to data items by the insider. These data items might be risky because they can be aggregated along with other data items using dependencies to derive or deduce unauthorized information [15,16]. For instance, assume that a relational database that contains academic staff information has the following dependency \{Rank→Salary\}. In addition, suppose that the data items (Name, Rank) or (Rank, Salary) are insensitive information, whereas (Name, Salary) is sensitive. Suppose that an insider, say Bob, has accessed the information (Name, Rank) of an academic staff “Alice”, say (Alice, “Assistant Professor”). Now, suppose that Bob is requesting the information (Rank, Salary) of Alice. In this case, if Bob is granted his request, he can use the dependency to combine the information (Alice, Assistant Professor), which is in his knowledges, with (Assistant Professor, Salary) to obtain the sensitive information (Alice, Salary). Obviously, building and checking knowledges are necessary for insider threat detection and prevention. Detecting and preventing insider threat by investigating knowledges of insiders and dependencies among data items have been researched extensively by many researchers such as Farkas et al. [16] and Yaseen and Panda [12–14]. Interested readers can refer to their research for more details. It is important to mention here that (Rank, Salary) is a common knowledges, and one may not be given an access to Rank if she/he should not know Salary. However, there exist many such dependencies that are not this obvious. For simplicity, we have used (Rank, Salary) example in this paper.

In light of the previous discussion, knowledges play a major role in insider threat, and checking it is a major step in any methodology that tackles insider threat in this context. Building and checking knowledges of an insider are not a hard problem in relational databases that are local in nature. It is built by profiling the insider’s accesses to data items and used when necessary to prevent threat. However, building knowledges in cloud relational databases and using it need new methodologies
because cloud databases are replicated in many sites to assure availability and reliability. The next section explains this problem in more details.

### 3.2. Insider threat in cloud relational databases

Cloud providers store data in multiple data centers that are both geographically and logically separated. A data center consists of connected servers and storage systems. Storage systems are aggregated into storage pools to form logical storage, which can be accessed from different computer systems that share the storage pool. A key benefit of this is that the data can be replicated or moved to other locations (storage locations) transparently to servers using it [32]. Availability zones in each data center are connected via inexpensive and low latency network. To achieve greater performance and fault tolerance, an application’s traffic may be distributed across multiple availability zones and data centers, which is called elastic load balancing [33]. Figure 1 shows the structure of Amazon cloud services. The figure shows that Amazon has five data centers across the globe. Each data center has more than one availability zone (AZ) [34].

Cloud relational databases are fragmented and replicated to increase the availability and reliability. The replication of data across availability zones and data centers should be consistent. Workloads on replicas’ nodes are balanced using live migration, where tenants (applications) are migrated from overloaded nodes to idle (or low-loaded) nodes to achieve load balancing. Users have no control on choosing the location or the instance that they prefer. Cloud systems choose the server, the location, and the storage that are needed for executing a process depending on some criteria such as the amount of load on servers or availability zones. Thus, different user’s requests may be executed on different instances in the same availability zone or in different availability zones or data centers.

Replication and load balancing increase the performance of cloud relational database systems. However, they may increase the probability of insider threat, which arises when a cloud relational database system fails to use the knowledgebase of insiders to detect the threat. In other words, an insider may combine data items she/he obtains from database instances in different availability zones to launch attacks. Figure 2 shows the problem. The insider accesses the data item D1 in the availability zone 1 (AZ1) (that may have the knowledgebase of the insider) and then accesses D2 in the availability zone n (AZn) (that may not have the insider’s knowledgebase). In this case, the system on the availability zone n fails to detect this threat and enables the insider to access D2. Thus, the insider combines the two data items and obtains the sensitive information S1, which is a threat.

In addition, insiders are allowed to access the cloud from any site in the globe, which is one of the features that the cloud offers. Cloud systems connect insiders to the closest availability zone (if it is not overloaded) to execute their queries in order to achieve the best performance. This means that insiders may be connected to different availability zones when they travel and work from different sites. In this case, insiders may be able to launch attacks using the same scenario described in Figure 2. It is important to mention here that to the best of our knowledge, no research has discussed the threat of knowledgebase in cloud environment and no research has discussed how to manage knowledgebases in this new environment.

In light of the previous discussion, an up-to-date knowledgebase of an insider should be checked at each access by the insider to prevent threats. Furthermore, the knowledgebase should be updated after each access the insider performs. Thus, cloud RDBMS need new methodologies to build, store, and synchronize knowledgebase in cloud environments because local knowledgebases are no longer suitable.

![Figure 1. Amazon’s cloud structure.](image-url)
4. MODELS FOR MITIGATING THE THREAT OF INSIDERS’ KNOWLEDGBASES

Securing cloud RDBMS against insider threat needs a methodology that monitors the activities of insiders in different instances and locations of cloud relational databases. The knowledgebases of insiders should be monitored and synchronized to achieve this purpose. In traditional RDBMS, building, maintaining, and checking knowledgebases are the responsibilities of organizations (owners). Nonetheless, when moving to the cloud, these operations are transferred to cloud providers (Cloud RDBMS). Keeping these responsibilities for local systems when moving to the cloud violates the concept of cloud computing. Moreover, keeping the knowledgebase of an insider in local storage needs transferring it with every access by the insider, which is infeasible because of the network overhead that it poses especially when knowledgebase obtains large. This section introduces four frameworks to maintain knowledgebases in a Cloud RDBMS and demonstrates the features and limitations of each one.

4.1. Peer-to-peer model

In this model, the knowledgebase of each insider is built and stored in all availability zones. At each access of an insider to a data item in an AZ, the knowledgebase of the insider in the availability zone is updated. Next, the updates are sent to all other availability zones and data centers simultaneously to keep knowledgebases consistent. Transactions are monitored locally at each availability zone or database instance. Thus, insider threat monitoring is performed locally without a need to communicate with other nodes. Figure 3 shows the proposed framework, where AZ denotes an availability zone, LB denotes load balancing and U(KBs) denotes updating of knowledgebases.

As shown in Figure 3, an insider sends his/her query to a cloud RDBMS. The cloud system sends the query to the closest AZ. If the AZ has a high load, the query is transferred to another AZ. In both cases, the insider’s knowledgebase is checked to ensure that there is no threat. Once the query is executed, the knowledgebase of the insider is updated, and the knowledgebases in all other availability zones are updated as well.

A key benefit of this model is that there is no single point of failure. Moreover, transactions and threat detection are executed fast because all processing are performed within a single availability zone and no communications are needed with other parts of the cloud system. In addition, the manipulating of knowledgebases is distributed among all availability zones, which balances their load.

The challenge that arises when using this model is the profiling of activities (building knowledgebases) for each insider. Local profiling is faster in processing transactions, but it imposes synchronization problems. Knowledgebases in all database servers should be updated simultaneously. Otherwise, insiders may access different data items in different sites (because of load balancing) and combine them using dependencies to pose a threat as discussed earlier.

Keeping knowledgebases updated needs a lot of immediate processing, which is time and resources consuming, and it causes delays in processing transactions. In summary, using this model poses high network traffic and delays transactions processing especially in the case of large number of replicas. Therefore, this approach is suitable when the number of instances is small.

To enhance the performance of this model, updating knowledgebases in some availability zones can be postponed when the processing load or network traffic is high. In this case, new processing requests by insiders should be distributed among up-to-date AZs only. Other AZs can be updated when traffic is low. This helps in increasing transactions’ processing performance and decreasing the delay that may be caused in case of high network traffic.

4.2. Centralized model

This model uses a coordinator site that builds, stores, and manages the knowledgebases of all insiders. Furthermore, the query of an insider is sent to the coordinator first. Then, the coordinator inspects the query against insider threat using the insider’s knowledgebase, which the coordinator has. If there is no threat found, the coordinator transmits the query to one of the cloud RDBMS nodes.
Mitigating insider threat in cloud relational databases

Q. Yaseen et al.

Figure 4. Centralized model.

(in an availability zone), considering the load balancing. After executing the query successfully, the cloud RDBMS send back an acknowledgement to the coordinator so that it updates the knowledgebase of the insider. The model in this state has a bottleneck. Strictly speaking, if the coordinator becomes unavailable because of some reason, the entire system of insider threat prediction and prevention fails. Figure 4 shows the modified model. The modified model uses a secondary coordinator to mitigate the bottleneck problem, which is similar to the idea used in damage recovery in distributed systems by Panda and Zue [35]. The secondary coordinator is used only in case of failure. However, the secondary knowledgebase should be updated to keep both knowledgebases consistent as shown in Figure 4, where $U(KBs)$ indicates updating the knowledgebases, $LB$ indicates load balancing, and $Ackn$ denotes an acknowledgement.

The advantages of this model include the relatively small amount of network traffic compared with the previous model. Thus, this model is more scalable than peer-to-peer model. Moreover, the synchronization is performed among the instances of knowledgebases only (the primary and secondary sites). That is, there is no delay occurring because of the synchronizing process among knowledgebases instances. However, the delay may happen because all requests are inspected and filtered at the central unit. Therefore, the central unit should be equipped with high performance capabilities to serve this job.

4.3. Mobile-knowledgebases model

This model has the advantages of peer-to-peer model and mitigates its disadvantages. In this model, an AZ in a data center stores the knowledgebases of insiders who are geographically close to it instead of storing the knowledgebases of all insiders. For example, Figure 5 shows how knowledgebases of insiders in the USA may be stored, where Arkansas insiders’ knowledgebases can be stored in AZ4 and Washington insiders’ knowledgebases can be stored in AZ1. Hence, the AZs may belong to different data centers. This model depends on the assumption that insiders are highly probably performing most of their work in one location (i.e., a company complex). However, an insider may perform his/her work from different (geographically) locations, which is a key advantage of cloud computing. In this case, the cloud system should send a copy of the knowledgebase of the insider to the new location to check his/her queries against insider threat. In the figure, Send $KBs$ stands for sending a copy of a knowledgebase of an insider, which may be needed when balancing a load or when an insider accesses an AZ that does not have his/her knowledgebase.

To show how the model works, suppose that an insider, say Bob, works for a company in Arkansas, which belongs to AZ4. Assume that Bob traveled to Washington, which belongs to AZ1, and he wants to perform some work for his company. Figure 6 shows how the model works in this case. Bob sends his query to the cloud system, which forwards his request to AZ1. The cloud system in this AZ checks whether Bob’s knowledgebase exists or not.

Figure 5. An example of Mobile-Knowledgebases model.

Figure 6. Executing queries in a Mobile-Knowledgebases model.
Because the knowledgebase is not available, the cloud system in AZ1 reaches other AZs asking for Bob’s knowledgebase. AZ4, which has the knowledgebase, responds and sends back a copy of Bob’s knowledgebase to AZ1. Then, the cloud system in AZ1 checks whether there is a threat posed by Bob. If there is no threat, AZ1 runs Bob’s request and sends the updates on the Bob’s knowledgebase to AZ4. Algorithm 1 shows how this model works in details.

Mobile-Knowledgebases model does not need to store the knowledgebases of all insiders in every availability zone as in peer-to-peer model. Moreover, it has less traffic than peer-to-peer model because the updates of knowledgebases are sent to host AZs only in case of “moving” insiders. Therefore, the Mobile-Knowledgebases model is more scalable than peer-to-peer model. Furthermore, the failure of an availability zone does not affect the queries of other insiders in other AZs. That is, it does not have a bottleneck as in the centralized model (more reliable). In addition, in most cases, the model needs to process the transactions and manage the knowledgebases of some insiders only, which means it has less processing overhead than other models.

This model can be further optimized in order to eliminate the need of sending messages to all AZs looking for the knowledgebase of a “moving insider”. This can be achieved by using a directory that stores the hosting AZ address of all insiders at an organization. Therefore, when an insider’s query is sent to an AZ other than his/her hosting one, the cloud system at the new AZ looks up the directory it has to retrieve the insider’s hosting AZ. Next, a message is sent to the latter AZ only to retrieve the knowl- edgebase of the insider. Keeping a directory for all insiders needs more storage, but it greatly minimizes the network traffic overhead, especially when the number of AZs and data centers becomes larger. We call this enhanced version Guided Mobile-Knowledgebases Model. Figure 7 shows the model.

Figure 7. Executing queries in the Guided Mobile-Knowledgebases model.
5. MANAGING DEPENDENCY GRAPHS AND THE LIFETIMES OF DATA ITEMS

Knowledgebases and dependency graphs are major parts in insider threat prediction and prevention models. We suggest using the dependency graphs proposed by Yaseen and Panda [12,13], which are Neural Dependency and Inference Graph (NDIG) and Constraint and Dependency Graph (CDG), in insider threat mitigation models in cloud RDBMS. Figure 8 shows an example of NDIG; interested readers can refer to the aforementioned research for more details about dependency graphs in relational database systems. In traditional insider threat mitigation models (in traditional RDBMS), dependency graphs are stored locally as a part of the models. In cloud RDBMS, the location of NDIG and constraint and dependency graph depends on which model we would adopt to manage knowledgebases. In peer-to-peer and Mobile-Knowledgebases models, dependency graphs should be stored in each AZ because insider threat prediction and prevention are performed at each one. However, in centralized model, dependency graphs need to be stored on the coordinators sites only. Checking the lifetimes of data items in knowledgebases is crucial. Not all data items in knowledgebases can be used for inference; only up-to-date data items are risky ones. For instance, recall the example mentioned before about the dependency (Rank → Salary). Suppose that Alice has read the (Name, Rank) of Professor Jim, say (Jim, Assistant Professor), which adds this data item to Alice knowledgebase. Now, if Alice obtains an access to the data item (Assistant Professor, Salary), she will infer the Salary of Jim, which is a threat. Thus, the system prevents her from accessing the latter data item. However, assume that another insider updated the Rank of Jim after Alice had accessed it. In this case, we say that the lifetime of the data item (Jim, Assistant Professor) has expired because the inference based on it will be incorrect. Thus, the data item (Assistant Professor, Salary) can be granted to Alice if she requests it because no threat could be posed in this case. We should mention here that the lifetime of a data item is expired if the inference based on its value is incorrect. Not every update process leads to lifetime expiration. The lifetimes of data items and updates were discussed in [16], and the conditions of considering a data item as expired or not due to an update process were discussed in [13,15]. Interested readers can refer to the previous research for more details about this issue. Clearly, checking the lifetimes of data items is crucial in insider threat mitigation. Checking knowledgebases and dependencies without checking the lifetimes of data items may limit the availability when no threat could be posed.

In light of the previous discussion, knowledgebases in cloud RDBMS should be managed with taking into account the lifetimes of data items such that expired data items are marked or deleted. Managing the lifetimes of data items in cloud RDBMS depends on the model used for managing knowledgebases. Two possible ways can be used to manage the lifetimes of data items cloud RDBMS, which are Exhaustive-Updating and Updating-on-Use.

(1) Exhaustive-Updating Approach: In this approach, at each write access of a data item by an insider, all knowledgebases of insiders are investigated searching for the data item. If the data item exists in one of knowledgebases, the value of the data item is checked against expiration. If the value is expired, the data item is either deleted or marked as expired. After completing this process, all instances of affected knowledgebases should be updated. Notice that, in this approach, a threat prediction model needs to investigate knowledgebase only to search for a risky data item and to check whether its lifetime is expired or not.

Using this approach in peer-to-peer model is time consuming and causes network traffic and processing overhead because the peer-to-peer model maintains knowledgebases at each availability zone. Once a knowledgebase is updated, the updates should be sent through networks to other cloud RDBMS nodes. Therefore, this approach can be used in small systems that have small number of insiders and data items and when the number of cloud RDBMS instances is small.

In centralized model, updating knowledgebases when using the Exhaustive-Updating approach is performed on the coordinators site only. Moreover, the network traffic occurs between the primary site and the secondary site only. Thus, less network traffic is resulted in comparing with the previous model, which means that it is more scalable. However, because all the processing of threat prediction and prevention is performed at coordinators site, using this approach adds more load to the coordinators node, which may overload it. Therefore, powerful capabilities should be guaranteed and maintained at the node.

The workload of using this approach in Mobile-Knowledgebases model is distributed among availability zones. Clearly, because the knowledgebases of a group of insiders are stored in the closest availability zone, updating a knowledgebase is performed locally, and no update is sent out through

---

Figure 8. Part of the dependency and inference graph of faculty staff database.
networks. That is, no network overhead is resulted as in peer-to-peer model, and contrarily to the centralized model, the processing overhead of maintaining knowledgebases is distributed among all availability zones. Thus, the best performance of the Exhaustive-Updating approach is achieved when it used with Mobile-Knowledgebases model.

(2) Updating-on-Use Approach: Contrarily to the Exhaustive-Updating approach, this method does not update knowledgebases immediately after each write operation. Instead, the lifetime of a data item (in a knowledgebase) is updated when it is checked against insider threat only and found expired. This process is performed as follows. At each read access to a data item by an insider, say Bob, if the data item can be used with another data item, say $F$, in Bob’s knowledgebase to pose a threat, the timestamp of $F$ in Bob’s knowledgebase is compared with the write timestamp of $F$ in the cloud RDBMS. If $F$ was updated after the last access to it by Bob, $F$ is called $P$-Expired, which indicates Possibly Expired. Next, the value of $F$ is investigated to check whether $F$ is expired or not. If it is expired, the data item is removed from Bob’s knowledgebase or marked as expired. This two-phase checking process eliminates the need to check the value of $F$ (to check it against expiration) in case it has not been overwritten after the last access by Bob. Strictly speaking, if $F$ has not been overwritten, there is no need to go further and check its value because it is not expired in this case. However, if $F$ has been updated, it might be expired ($P$-Expired). Therefore, $F$’s value needs to be checked to ensure that it is expired.

Obviously, using this approach reduces the processing overhead needed to investigate the knowledgebases and update them at each write access. However, it adds more processing time to transactions because it needs checking both knowledgebases and cloud RDBMS in order to check the lifetimes of data items during transactions processing.

Adopting this approach in peer-to-peer model does not pose great network traffic overhead because updates to knowledgebases are sent gradually (when an expired data item is discovered), which is greatly less than the overhead that is posed when using the Exhaustive-Updating approach. However, the processing time needed for transactions is greater than that needed in the Exhaustive-Updating approach as mentioned earlier. Similarly, in centralized model, using this approach reduces the network traffic between the primary and secondary coordinators in comparing with the Exhaustive-Updating approach but adds more processing time to transactions. In Mobile-Knowledgebases model, no extra network traffic is resulted as in the Exhaustive-Updating approach. However, similarly to the other models, more processing time is needed for transactions.

6. EXPERIMENTS AND ANALYSIS

To compare between the proposed models, we used Cloud-Exp Simulator [18], which is an extended version of the CloudSim Simulator. The host specifications are fully described in Table I, where every host has one virtual machine. In addition, Table II shows the properties of the AZs used in the simulations.

The simulations were conducted using five AZs and 50 data items. In the case of Mobile-Knowledgebase and Guided Mobile-Knowledgebase models, the percentage of requests that were sent to non-hosting availability zones (the availability zones that do not have the knowledgebase of the requesting insiders) was set to 5%. The workload was randomized for all the models. For the rest of the paper, the abbreviations P2P, CENT, MOBILE, and G-MOBILE indicates peer-to-peer model, centralized model, Mobile-Knowledgebases model, and Guided Mobile-Knowledgebases model, respectively. In addition, the HOST-AZ indicates the AZ that has the knowledgebase of the requesting insider.

6.1. Variable number of queries per insider

This simulation was performed to show the effect of the number of queries sent by insiders on the performance of the system and the network in each model. In this simulation, the percentage of accessible data items per insider, which are the data items that the insider is allowed to access, was set to 80%, and the percentage of dependency among data items was set to 9%. The number of insiders was set to 100 insiders.

Table I. Host properties.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host ID</td>
<td>ID (0- number of hosts)</td>
</tr>
<tr>
<td>Storage</td>
<td>1 TB</td>
</tr>
<tr>
<td>MIPS for CPU</td>
<td>1024</td>
</tr>
<tr>
<td>Memory capacity</td>
<td>2 GB</td>
</tr>
<tr>
<td>Network bandwidth</td>
<td>10 Mbps</td>
</tr>
<tr>
<td>Virtual machine scheduler</td>
<td>Space shared</td>
</tr>
</tbody>
</table>

CPU, central processing unit; MIPS, million instructions per second.

Table II. Availability zone properties.

<table>
<thead>
<tr>
<th>Part (MB)</th>
<th>Availability Zone property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ram</td>
<td>2048</td>
</tr>
<tr>
<td>CPU(MIPS)</td>
<td>1000 single core</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>10,000</td>
</tr>
<tr>
<td>Storage(MB)</td>
<td>1,000,000</td>
</tr>
</tbody>
</table>

MIPS, million instructions per second.
6.1.1. Network overhead.

Figure 9 demonstrates the effect of the number of queries per insider on the network overhead when using the four models, which is measured by the number of packets sent through the network. Basically, increasing the number of queries, which an insider sends, increases the network overhead. However, the effect of increasing the number of queries is the largest when using the P2P model, and the overhead is the smallest when using the G-MOBILE model. As seen in Figure 9, the overhead that the G-MOBILE model poses is very small in comparing with other models.

In P2P model, when the number of queries increases, the synchronization messages that are sent among availability zones increase. This leads to a great overhead in the P2P model as shown in the figure. In CENT model, the synchronization messages are sent between the primary and secondary units, and between the primary unit and one AZ only for a specific request. Therefore, the number of messages sent to network is less than those sent when using the P2P model. In the case of MOBILE model, when increasing the number of queries an insider requests, the probability that a request is sent to a distant AZ increases. This clarifies the result in the figure, which shows that the network overhead increases as the number of queries increases when using the MOBILE and the G-MOBILE models. However, the overhead is less than that posed by P2P and CENT models, because in MOBILE model the availability zones exchange messages when an AZ does not have the knowledgebase of the requesting insider. Obviously, the G-MOBILE model poses less overhead than MOBILE model because it does not broadcast requests searching for the Host-AZ as MOBILE model. Instead, it communicates a directory to find the HOST-AZ, which reduces the network overhead as the figure shows. One more observation to raise, the growth of MOBILE model is larger than the growth of the CENT model. This is clear when we increase the number of queries per insider to 30. Clearly, as discussed before, when increasing the number of requests per insider, more requests are sent to non-hosting AZ (not a HOST-AZ) because of load balancing. This process increases the number of exchanged messages between the availability zones that receive the request and the HOST-AZ, which increases the network overhead because of transferring the knowledgebase of the requesting insider and the updates that are sent back to the HOST-AZ. This clarifies why the growth in MOBILE model is greater than the growth in CENT model.

6.1.2. System Performance.

Figure 10 shows the effect of increasing the number of queries that are sent by insiders on the system performance. As shown in the figure, the processing time needed to finish all workload when using the CENT model increases greatly as the number of queries increases. However, the processing time is very close to other models.
The CENT model has a processing unit that deals with all insiders requests and performs all insider threat detection. Clearly, as the load increases, the time needed to serve requests increases because of delay. However, in other models, the requests are distributed among all availability zones. Therefore, a small increase in the number of requests per insider that were performed in this simulation does not affect the performance of the processing units. The simulation in this part shows that the G-MOBILE model has the best performance when the number of queries is large in the cloud system.

6.2. Variable number of insiders

This simulation was conducted to demonstrate the effect of the number of insiders on the performance of the system and the network. In this simulation, the percentage of accessible data items per insider, which are the data items that the insider is allowed to access, was set to 80%, and the percentage of dependency among data items was set to 9%. The number of insiders was set to 100 insiders.

6.2.1. Network overhead.

This simulation shows how increasing the number of insiders affects the network when using the proposed models. Figure 11 shows that the G-MOBILE model has the least effect on the network when increasing the number of insiders. Meanwhile, the P2P model has the largest network overhead as the number of insiders increases.

Increasing the number of insiders means more knowledgebases that have to be stored. In case of P2P, all the knowledgebases have to be stored and synchronized in all AZs, which explains the huge impact on the network overhead as the number of knowledgebases increases. However, when using G-MOBILE model, knowledgebases are stored locally and are not synchronized in other locations. Furthermore, because only 5% of requests are sent to non-hosting AZs (not a HOST-AZ), few messages are sent to the network. In MOBILE model, it has more traffic than G-MOBILE model because broadcast messages are sent searching for the HOST-AZ. In P2P model, the added traffic is medium because the synchronization is performed between the primary and the secondary units only. In addition, updates on knowledgebases are sent back to the primary unit through network.

6.2.2. System overhead.

This simulation shows the effect of increasing the number of insiders on the system performance. Figure 12 demonstrates similar results to what we discussed in Figure 11. Obviously, the most affected model is the CENT model because it is increasingly overloaded as the number of insiders increases. Therefore, the delay in serving requests increases as shown in the figure. However, the increase in processing delay when using other models is small. As discussed before, the requests are distributed among all AZs because the increase in insiders is distributed in all AZs. Therefore, a small

Figure 11. Number of insiders versus network overhead. P2P, peer-to-peer model; CENT, centralized model; MOBILE, Mobile-Knowledgebases model; G-MOBILE, Guided Mobile-Knowledgebases model.

Figure 12. Number of insiders versus system overhead. P2P, peer-to-peer model; CENT, centralized model; MOBILE, Mobile-Knowledgebases model; G-MOBILE, Guided Mobile-Knowledgebases model.
increase in the processing time is added as the number of insiders increase.

This simulation proves that increasing the number of insiders has a little effect on the G-MOBILE model and it is the best model to use when the cloud system has very large number of insiders.

6.3. Variable percentage of accessible data items

Accessing more data items by insiders increases the size of their knowledgebases, which in turn, increases the synchronization process among replicas when exist. Therefore, we conducted this simulation to show to what extent the aforementioned argument is correct. In this simulation, the percentage of dependency among data items was set to 9%, and the number of insiders was set to 100 insiders.

Figure 13 demonstrates the effect of increasing the percentage of accessible data items. As shown in the figure, increasing the allowable data items has a great effect on the P2P model and has a considerable effect on the CENT model. That is, increasing the percentage of accessible data items increases the network overhead in P2P and CENT models. However, the effect is negligible on the MOBILE and G-MOBILE models.

The P2P model synchronizes the knowledgebases in all availability zones. Therefore, increasing the accessible data items expands the knowledgebases of insiders, and as a result, it increases the synchronization process. The aforementioned process greatly maximizes the network overhead as shown by the figure. However, in the centralized model, the synchronization is performed between the primary and the secondary units, which is much less than the synchronization in the P2P model. Therefore, the growth in the network overhead when using the CENT model is greatly less than the growth in the P2P model. In MOBILE and G-MOBILE models, the size of knowledgebases has a negative effect on the network overhead when the receiving AZ does not have the knowledgebase of the requesting insider (not a HOST-AZ). In this case, transferring large knowledgebases increases the network overhead. Nonetheless, the aforementioned process occurs in 5% of the cases, which explains the negligible growth in the network overhead when using those models. Hence, the growth in the network overhead will be greater when requests are largely sent to non-hosting AZs (not HOST-AZs).

In light of the previous discussion, using the G-MOBILE model has the least network overhead among all models despite the percentage of accessible data items.

7. CONCLUSIONS

Cloud security is among the major issues that worry both individuals and organizations when moving data to the cloud. One of the riskiest issues is the insider threat because of the extreme harm that it may pose. This paper has investigated insider threat in cloud relational databases. It has demonstrated how workload balancing across AZs and data centers may help insiders to bypass security mechanisms and launch attacks. To mitigate insider threat, the paper has proposed new insider threat prediction and mitigation models that are suitable for the cloud environment, which are peer-to-peer model, centralized model, Mobile-Knowledgebases model, and Guided Mobile-Knowledgebases model. In these models, the paper has shown how to manage and synchronize knowledgebases, dependency graphs, and updates on data items to defend cloud RDBMS against insider threat. The paper has tested the proposed models to show their effectiveness under different environments such as large number of insiders, large number of queries, and large number of accessible data items. The experiments have compared among the proposed models regarding the overhead that they may add to the network and processing systems. In addition, the experiments have shown that the Guided Mobile-Knowledgebase model has the best performance.

REFERENCES


15. Yaseen Q, Panda B. Organizing access privileges: maximizing the availability and mitigating the threat of insiders knowledgebase. Proceedings NSS'2010, Melbourne, Australia, 2010; 312–317.


