Satisfying Least Privilege through Database Decomposition

Fabrizio Baiardi
Università Di Pisa
56127 Pisa
fabrizio.baiardi@unipi.it
ORCID: 0000-0001-9797-2380

Cosimo Comella
Autorità GPDP
00187 Roma
c.comella@gpdp.it

Vincenzo Sammartino
Università Di Pisa
56127 Pisa
v.sammartino@studenti.unipi.it
ORCID: 0009-0002-4632-1179

***Abstract*—The Multilevel Database Decomposition Framework aims to enhance the robustness and minimize data leakage of intrusions into healthcare databases. To this purpose, the framework decomposes a database into smaller ones to restrict user access according to the least privilege principle. Each database this decomposition returns is uniquely associated with a distinct set of users so that each user can access all and only the data his/her operations need. Furthermore, the framework minimize the impact of impersonation attacks by confining an intrusion to just one of the databases the decomposition returns. This is achieved by mapping the databases the decomposition returns onto distinct virtual or physical machines according to the robustness of the confinement to be achieved. Beside increasing robustness, this allocation reinforces defenses against evolving cyber threats.**

As a counterpart of better robustness, the decomposition replicates some tables across the databases it returns. To prevent possible inconsistencies updates of a table are spread to its copies in distinct databases. We present a performance analysis to evaluate the overhead of each allocation due to multiple copy updates. The analysis supports a fine-tuning of the final performance of alternative database allocation to the resulting robustness. We exemplify the application of the framework to a simple healthcare database.

I. Introduction

The Multilevel Database Decomposition Framework (MDDF) is a general solution for the protection of personal information but with a focus on healthcare applications. It aims to improve the robustness of applications that exploit relational databases against impersonation attacks where an intruder impersonates a legal user. The key strategy of MDDF is to fully satisfy the least privilege principle to effectively mitigate the data leak due to impersonations and safeguard healthcare information [6, 5].

The MDDF key strategy is decomposing one relational database into a set of databases defined according to the operations of each user. This minimizes user access rights [11, 16] because each user can access a database with all and only the data his/her operations need. MDDF strongly improves the overall data security by reducing the blast radius of a successful intrusion that leaks some data due to impersonation attacks [3]. In these attacks a threat agent impersonates a legitimate user after stealing the user credentials.

To prevent the spread of intrusions across the databases the decomposition returns, MDDF supports alternative allocation strategies of these databases. While the simplest allocation maps all databases onto the same machine, confinement and robustness are enhanced by distributing databases across distinct containers or physical/virtual machines. The ability to choose the allocation according to the robustness of interest is a fundamental advantage that MDDF offers with respect to the simple management of the user access rights on the resulting databases.

The databases that MDDF produces may share tables or subsets of tables when some users only access some attributes of a table. Shared tables are replicated across databases, and updates are replicated to maintain consistency among the copies of a shared table and to offer the same data view to each user. Multiple copy update introduces both complexity and overhead that are proportional to the robustness of the separation [10]. Another problem due to table replication is the larger amount of memory utilization. We believe this issue may be neglected when considering the more robust separation among the databases and the ability to use one copy to restore consistent information after any intrusion that maliciously updates a table. From this perspective, the MDDF uses less memory than solutions based upon distributed ledgers [1].

The original contributions of this paper are the definition of the strategy to decompose a database into several ones and of alternative mappings of these databases onto containers, virtual machines and physical nodes to minimize the probability an intrusion spreads among the databases the decomposition returns. In the following, we consider SQL databases but the proposed methodology can be applied also to NoSQL databases.

The paper is structured as follows: Sect. 2 briefly reviews related works, the least privilege principle, and other concepts underlying the MDDF. Sect. 3 describes the MDDF, emphasizing database decomposition. It also discusses how the allocation of the resulting databases can minimize the spreading of an intrusion. Sect. 4 analyzes the multiple copy update overhead as a function of the chosen allocation. Lastly, Sect. 5 exemplifies an application of MDDF.

II. State of the Art

The principle of least privilege (PoLP) suggests minimizing the user access rights so that no user owns access rights he/she does not need. Several strategies exist to satisfy this principle ranging from minimization of user access rights, to avoid hierarchical level of privileges and to network segmentation and defence in depth [14]. We are interested in adopting the PoLP to minimize the impact of an intrusion where a threat agent impersonates a legal user. State-of-the-art strategies to face the challenges posed by evolving cyber threats and cloud-based solutions improve database security by integrating the satisfaction of the PoLP with role-based access control (RBAC), encryption, tokenization, dynamic data masking, and pseudonymization [12, 7].

Role-Based Access Control (RBAC) [15] complements PoLP by assigning access permissions based on predefined roles. This grants access rights to each user according to the specific role of the user in an organization.

Encryption and **Tokenization** are crucial components in securing sensitive data [8] that, respectively, transform data into unreadable formats and replace sensitive data with non-sensitive placeholders.

Dynamic Data Masking (DDM) [4] obfuscates sensitive information dynamically to ensure that only authorized individuals can access and view sensitive data.

Adaptive and comprehensive security measures are essential to counter the evolving **cyber threat landscape** of databases that includes challenges such as ransomware, advanced persistent threats (APTs), and insider threats [9].

The widespread adoption of **cloud-based database** solutions introduces new security concerns [19] because protecting data in a cloud requires guarding against unauthorized access, data breaches from users of the same provider, and compliance with data residency regulations.

Pseudonymization is another vital facet of database security as it replaces personally identifiable information (PII) with artificial identifiers or pseudonyms [2]. This adds another layer of privacy protection by making it challenging to directly associate sensitive data with specific individuals. Pseudonymization enhances data security by both reducing the impact of unauthorized access to personal information and limiting exposure in the event of a breach.

RBAC and all the previous mechanisms that update or replace some information in a database can be easily integrated with the MDDF. This also holds for cloud-based databases because a cloud architecture strongly reduces the cost of allocating to distinct virtual machines the databases that MDDF produces.

III. Multilevel Database Decomposition Framework

This section details the decomposition step by step to outline the underlying principles. To this end, it is essential to explain how to decompose a database and then move from the decomposition of a database to that of its tables. Then, the section highlights the synchronization problems generated by a decomposition where the same table is shared, i.e. it belongs to distinct databases. This requires that some operations on a shared table in a database fire an automatic update of distinct databases the decomposition produces and that share the same table.

A. General Approach

MDDF aims to fully satisfy the PoLP [11] by decomposing a relational database shared among a set of users to minimize user access to the data involved in the operations each user can invoke. Starting from a system where each user can access any information in the database, first of all, the framework partitions users into groups, ie equivalence classes, where two users belong to the same group if they can invoke the same database operations. An operation reads or updates some attributes of some database tables. Then, MDDF decomposes the original database into distinct ones, each a subset of the original one and with the smallest amount of data to implement the operations of a class of users. The decomposition distributes the original tables to the subsets so that each subset includes all the tables to implement the operations of a class of users. If these operations do not access an attribute of a table, this attribute is dropped from the database corresponding to the class. Each user is assigned access rights on the tables in just one of the databases the decomposition returns, the one paired with the user class. This strategy satisfies the PoLP as it assigns all and only the access rights the user needs. To integrate the MDDF with any of the mechanisms discussed in the previous section we simply apply the mechanism either to the original ones or to those the MDDF returns.

The last step of the MDDF maps the databases it builds to distinct containers, virtual or physical machines to confine a successful intrusion to one of the databases.

Definition and Purpose: We assume the original database satisfies the third normal forms, 3NF and decompose it otherwise. This first decomposition aims to reduce redundancy, improve maintainability, and enhance the overall database performance.

The database DB is decomposed through a sequence of steps that, given a set of users U, can be described as follows:

- DB Normalization in 3NF
- Identification of Groups of U
- Creation of DB Subsets
- Association of Groups and Subsets
- Configuring Access Mechanisms
- Choosing the Confinement Robustness

B. Steps

The following subsections will show in detail the steps to be followed to apply the framework.

1) Traditional Normalization in 3NF: Normalization structures data in a database to eliminate redundancy and dependency. The third Normal Form (3NF) is a widely used standard that reduces data redundancy. We apply this normalization to ensure that DB is well structured and data is stored consistently. Furthermore, the third normal normalization can reduces the data in tables that are shared among DB subsets.

2) Identification of User Groups: Identifying users is a preliminary step that identifies all users who interact with the database and analyzes the needs of the operations of each user. In this way, we create groups of users tailored to their needs because two users belong to the same group if they invoke the same operations on the same tables. This limits access to sensitive information only to users who are entitled to operate on such information according to the *need-to-know principle*.

3) Creation of Database Subsets: The creation of multiple databases, each a subset of DB is the central step of the framework. It creates a distinct database for each user group in U, according to the operation the users in the group execute and the data these operations access. We consider the worst case, e.g. any table and any attribute a user may require should belong to the corresponding subset to ensure that users can access all and only the information they need. As a consequence, some tables will be shared among users in distinct groups and appear in distinct databases. The attributes of these tables implement data exchange among users in distinct groups. This requires that an update in a table is spread across all the databases where the table, or the attribute, appears.

4) Association of Groups and Subsets: The strategy to decompose DB implies a biunivocal association between user groups and the databases the decomposition returns. The association is the input to define user access rights. The membership in a group is dynamic because it depends upon roles an organization assigns.

5) Configuring the Access Mechanism: This step offers a first level of security by granting each user the access rights to access all and only the information in the database associated with the group the user belongs to. This prevents users from accessing information in DB that their work does not need. The detailed implementation of this step depends upon both the database management system adopted for DB and the underlying operating system. The restriction of each user to just one database minimises the blast radius of an intrusion i.e. the amount of data that may be lost because of data breaches and/or an unauthorized access.

6) Choosing the Confinement Robustness: The allocation of the databases from the previous steps onto physical machines, virtual machines, VMs, or containers determines the separation among the databases. This level is chosen according to the robustness of the overall system to protect the various information. The choice of allocation usually depends on the protection of the information in a database and on the level of confinement of an intrusion or an impersonation. In general, the more critical the information in a database, the more robust its confinement.

C. Confinement

MDDF enables the designer to select distinct allocations with increasing confinement levels to align the allocation with required security levels, resource usage, management complexity, and other factors. In more detail, each allocation offers a distinct robustness of the confinement that also depends upon vulnerabilities in containers or virtual machines [18, 17, 13]. Table 1 outlines the confinement levels, or robustness, resulting from alternative database allocations. Allocating databases to distinct physical machines provides the highest level of confinement, while a simple database decomposition offers the lowest.

Mapping	**Confinement Robustness**
Distinct physical machines	High
Distinct VMs	Medium-High
Distinct containers	Medium
Simple decomposition	Low

TABLE I
ROBUSTNESS LEVELS OF ALTERNATIVE MAPPINGS.

The selection of the proper allocation should consider that distinct physical machines offer high safety and confinement due to physical separation but come with high resource usage and management complexity. Distinct VMs provide good confinement with customizable resource allocation, a lower resource overhead and require expertise for management. Distinct containers are a lightweight solution with minimal overhead, but a larger risk due to the shared OS. A simple database decomposition is the easiest to manage and requires the fewest resources, but it results in the worst confinement and potential data integrity issues.

Even if an intrusion can successfully attack some of the previous allocations using vulnerabilities in virtual machines or containers, all these allocations offer better robustness to intrusions than the decomposition produced by the 3NF.

The choice of the proper allocation should align with the requirements of the system of interest, balancing confinement, resource efficiency, and management complexity. As an example, if DB1, DB2, DB3 are the databases the MDDF returns and if the information in DB1 is the most critical one, then we can map DB1 onto a physical machine and the two other databases onto a distinct one or two VMs mapped on the same physical machine. If the databases store information with distinct, but not too far, critical levels, they can be allocated to distinct VMs on the same physical machine. A cloud architecture which usually offers a large number of VMs mapped onto the same physical nodes or onto distinct ones can exploit at best the freedom of allocation to optimize the cost/performance ratio.

We have implemented some experiments to evaluate the overhead resulting from each of the previous allocations. According to our results, a multiple update takes about 70 ms when the databases are mapped onto distinct containers running on the same machine. The same update takes 280 ms if the databases are allocated to distinct virtual machines running on the same physical machine. Lastly, the update takes 800 ms if databases are allocated onto distinct physical machines. Obviously, the overall overhead due to multiple updates is also related to the probability of updating a database vs the one of an interrogation of the databases.

IV. Comparison of Alternative Synchronization Solutions

This section compares the overhead of two solutions to synchronize tables shared among distinct database subsets. The first solution, Trigger-API, uses triggers, ie database events that execute code in response to specific conditions, ensuring data integrity. The second solution utilizes SymmetricDS, an open-source tool supporting the replication of distributed environments. OpenMRS uses SymmetricDS to develop software for healthcare delivery in developing countries.

The performances of the two solutions have been evaluated under various conditions through a set of experiments. The comparison shows that Trigger-API consistently outperforms SymmetricDS across various metrics. Trigger-API demonstrates lower latency in local, remote, and distributed configurations, as well as better CPU utilization under light and medium workloads. Additionally, Trigger-API excels in handling data conflicts and achieves faster two-way synchronization. However, SymmetricDS shows slightly better transaction rates during calm periods. Overall, Trigger-API exhibits a better rate than SymmetricDS across various scenarios.

When evaluating the latency to transmit a table update to another database, Trigger-API consistently outperforms SymmetricDS across all configurations, resulting in a lower latency in local, remote, and distributed setups. Trigger-API results in significantly lower CPU utilization than SymmetricDS, mainly under light and medium workloads. Trigger-API offers the best performance in handling data conflicts, ie updates of the same table, for all percentages of resolved conflicts. It also offers a better performance than SymmetricDS in two-way Synchronization, in both directions. As concerns the number of transactions that can be served, Trigger-API offers higher transaction rates during peak activity, while SymmetricDS shows slightly better performance with lower activity.

V. An Example

This example considers a healthcare organization, specifically a hospital, managing information about operators, patients, prescriptions, and related administration. Initially, the hospital uses a centralized database to record patient details, prescriptions, and other relevant data. The database management system ensures medical professionals have access to patient information, while patients can retrieve their medical records and prescription details. The system also includes statisticians who analyze aggregated data without compromising patient identities. Patient names are replaced with pseudonymous IDs for privacy, and a mapping table simplifies the association between these pseudonymous IDs and patients.

A. Users and Database

The Healthcare Database (HealthDB) serves Patients, Medical Doctors, Nurses, Administrative Staff, and Statisticians.

Users:

- **Patients (Patients)**: Read access to their medical records, prescription details, and exam results, facilitated through a pseudonymous ID.
- **Medical Doctors (Doctors)**: Read and write access to patient information, medical history, and the ability to prescribe medications and order exams.
- **Nurses (Nurses)**: Read and write access to patient information, medical history, and the ability to record exam results and administer prescribed medications.
- **Administrative Staff (Admins)**: Write access to the CostsTable for billing purposes.
- **Statisticians (Statisticians)**: Read-only access to aggregated data for statistical analysis.

Database:

- **Healthcare Database (HealthDB)**: Centralized database storing the following tables:
  - **Patient Information Table (Patient)**: Includes pseudonymous IDs (PatientID), and IDs of assigned doctors and nurses. A mapping table (MappingTable) links real patient identities and pseudonymous IDs.
  - **Sensitive Data Table (SensitiveData)**: Stores sensitive patient information such as real names, and addresses. Linked to Patient via pseudonymous IDs.
  - **Billing** Stores payer details and overall cost to be payed for each patient. Linked to PatientTable via pseudonymous IDs.
  - **Prescription Records Table (Prescription)**: It stores prescription and exam data for each patient and it is linked to Patient by pseudonymous IDs.
  - **Exam Records Table (Exam)**: Stores exam results.
  - **Costs Table (Costs)**: Manages the costs of medicines and exams.
  - **Mapping Table (MappingTable)**: Maps patientIDs into real identities.
  - **Statistics Table (Statistics)**: Stores aggregated and de-identified data for statisticians.

B. MDDF Implementation Steps

We can identify the following groups:

1) **Patients**: Read access to their own medical records, prescription details, and exam records,
2) **Doctors**: Read and write access to patient information, ability to prescribe medications and order exams.
3) **Nurses**: Read and write access to patient information, ability to record exam results and administer p
4) **Administrative Staff (Admins)**: Read and Write access to the prescriptions and costs for billing purpose
5) **Statisticians**: Read-only access to aggregated data for statistical analysis.

Starting from the operation of these groups and taking into account that Doctors and Nurses access, with distinct access rights, the same tables, the following subsets are defined:

- **Subset MedDB**: It includes Patient, SensitiveData, Prescription, and Exam. These tables are essential for medical professionals, Doctors and Nurses, to access patient information and medical history, and prescribe medications or order exams.

- **Subset PatientDB**: It includes Patient and Exam. We assume each patient knows his/her pseudonymous ID. The subset provides patients with read access to their medical records, prescription details, and exam results.
- **Subset AdminDB**: It includes Prescription, Costs, and MappingTable. The key table is CostsTable which manages the costs of medicines and exams.
- **Subset StatDB**: It includes Patient, Prescription, Cost and Statistics. Statistics stores aggregated and pseudo anonymized data from the Patients table.

The access rights of each group are:

- **Patients**: Read access to PatientDB.
- **Doctors**: Read and write access to MedDB, in particular read and write access to PatientTable, read access to all the other tables.
- **Nurses**: Read and write access to all the tables in MedB.
- **Admins**: Read access to AdminDB, in particular read-only access to PrescriptionTable and MappingTable, read and write access to CostsTable.
- **Statisticians**: Read-only access to PatientTable, PrescriptionTable and CostTable. Write access to StatisticTable.

The shared tables are: Patient, Prescription, Cost and MappingTable. The multiple update strategy should exploit that only Doctors and Nurses produce data that other groups consume. Hence, the most effective solution is Trigger-API where operations by Doctors and Nurses fire the transmission of the new attribute values to tables in other subsets.

To optimize the confinment robustness, PatientDB and MappingTable store the most critical information and should be mapped onto a dedicated physical machine. Other subsets are mapped onto a distinct machine and allocated to distinct VMs or containers.

VI. Conclusions

Important benefits of the MDDF are a reduced impact of security breaches and better privacy this is important because the healthcare sector is a prime target for cyber-attacks. The MDDF limits the impact of security breaches, as compromising one database does not expose records in distinct ones. Furthermore, the MDDF restricts access to only the data subset the user needs to ensure that only authorized users can access specific patient information according to GDPR [6] and HIPAA [5]. Further advantages include a lower complexity of database management that results in better scalability and an easier sharing of data for medical research among healthcare organizations and researchers that can access specific subsets, streamlining data retrieval for research purposes.

The adoption of the MDDF can simplify data structures. Access to smaller, more tailored tables may improve the overall performance. As collected information expands, the MDDF can improve scalability by accommodating the growing volume of patient data in new subsets.

As a counterpart, the MDDF increases the complexity of database administration and maintenance as it requires more resources. A further problem is the synchronization overhead to spread the updates and preserve the consistency of data in shared tables.

Acknowledgments

The work of Vincenzo Sammartino was supported by the PRA 2022/2023 Research Project.

References

[1] K. Ahmad et al. “Hyperledger Fabric Enabled Vaccine Intelligent Network to Implement Immunization Program”. In: *2023 IEEE 12th International Conference on Communication Systems and Network Technologies (CSNT)*. 2023, pp. 822–828.

[2] M. Binjubeir et al. “Comprehensive Survey on Big Data Privacy Protection”. In: *IEEE Access* 8 (2020), pp. 20067–20079.

[3] M. Campobasso and L. Allodi. “Impersonation-as-a-Service: Characterizing the Emerging Criminal Infrastructure for User Impersonation at Scale”. In: *Proc. of the IEEE Int. Symposium on Secure Software Engineering*. Vol. 1. IEEE. 2006, p. 1.

[4] A. Cuzzocrea and H. Shahriar. “Data masking techniques for NoSQL database security: A systematic review”. In: *2017 IEEE International Conference on Big Data (Big Data)*. 2017, pp. 4467–4473.

[5] L. A. Denise, A. Ajit, and J. M. Eric. “Institutionalizing HIPAA Compliance: Organizations and Competing Logics in U.S. Health Care”. In: *Journal of Health and Social Behavior* 55.1 (2014), pp. 108–124.

[6] European Union. *Data protection in the EU*. 2023.

[7] M. Humayun et al. “Security threat and vulnerability assessment and measurement in secure software development”. In: *Computers, Materials and Continua* 71 (2022), pp. 5039–5059.

[8] S. Ibrahim et al. “A novel data encryption algorithm to ensure database security”. In: *Acta Infologica* 7.1 (2023), pp. 1–16.

[9] I. Linkov et al. “Applying resilience to hybrid threats”. In: *IEEE Security & Privacy* 17.5 (2019), pp. 78–83.

[10] S. Ab. Moiz et al. “Database replication: A survey of open source and commercial tools”. In: *International Journal of Computer Applications* 13.6 (2011), pp. 1–8.

[11] J. H. Saltzer and M. D. Schroeder. “The Protection of Information in Computer Systems”. In: *Proc. IEEE* 63.9 (1975), pp. 1278–1308.

[12] N.A. Al-Sayid and D. Aldlaeen. “Database security threats: A survey study”. In: *2013 5th International Conference on Computer Science and Information Technology*. 2013, pp. 60–64.

[13] S. Shringarputale et al. “Co-residency Attacks on Containers are Real”. In: *Proc. of the 2020 ACM SIGSAC Conf. on Cloud Computing Security Workshop*. ACM. 2020, pp. 53–66.

[14] William R Simpson and Kevin E Foltz. “Network segmentation and zero trust architectures”. In: *Lecture Notes in Engineering and Computer Science, Proceedings of the World Congress on Engineering (WCE)*. 2021, pp. 201–206.

[15] I. Singh et al. “Database intrusion detection using role and user behavior based risk assessment”. In: *Journal of Information Security and Applications* 55 (2020), p. 102654.

[16] R. Smith. “A Contemporary Look at Saltzer and Schroeder’s 1975 Design Principles”. In: *IEEE Security & Privacy* 10.6 (2012), pp. 20–25.

[17] A. Y. Wong et al. “On the Security of Containers: Threat Modeling, Attack Analysis, and Mitigation Strategies”. In: *Computers & Security* 128 (2023), p. 103140.

[18] J. Wu et al. “An access control model for preventing virtual machine escape attack”. In: *Future Internet* 9.2 (2017), p. 20.

[19] P. Yang, N. Xiong, and J. Ren. “Data security and privacy protection for cloud storage: A survey”. In: *IEEE Access* 8 (2020), pp. 131723–131740.