An Efficient Solution to User Authorization Query Problem in RBAC Systems Using Hierarchical Clustering

K. Rajesh Rao[1], Aditya Kolpe[1], Tribikram Pradhan[1,*] and Bruno Bogaz Zarpelão[2]

[1]*Department of Information and Communication Technology, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal, Karnataka, India*
[2]*Department of Computer Science, State University of Londrina, Londrina-PR, Brazil*
E-mail: tribikram.pradhan@manipal.edu
[*]*Corresponding Author*

Received 23 April 2021; Accepted 26 December 2021;
Publication 07 November 2022

Abstract

Role Based Access Control (RBAC) systems face an essential issue related to systematic handling of users' access requests known as the User Authentication Query (UAQ) Problem. In this paper, we show that the UAQ problem can be resolved using Unsupervised machine learning following the guaranteed access request and Dynamic Separation of Duty relations. The use of Agglomerative Hierarchical Clustering not only improves efficiency but also avoids disordered merging of existing roles to create new ones and steers clear of duplication. With a time complexity of $O(n^3)$, the algorithm proves

Journal of Cyber Security and Mobility, Vol. 11_4, 531–548.
doi: 10.13052/jcsm2245-1439.1142

to be one of the fastest and promising models in state-of-the-art. The proposed model has been compared with the existing models and experimentally evaluated.

Keywords: Hierarchical agglomerative clustering, least privilege principle, role based access control, separation of duties, user authorization query problem.

1 Introduction

Role-based access control (RBAC) is an access control mechanism focused on associating roles to the individual users, proposed by [1]. In 2000, a unified model for RBAC by Ravi Sandhu et al. was published as a National Institute of Standards and Technology (NIST) RBAC model and in 2004, the model was adopted as an American National Standard for Information Technology (ANSI). The RBAC reference model defines a set of essential RBAC elements and the relation between them. The users, roles and permissions are the essential RBAC elements. The set of sessions is also added to the above elements, where each session is a mapping between an user and an activated subset of roles that are assigned to the user [2]. Further, research is also carried out by dynamically assigning the roles to user [3].

The User Authorization Query (UAQ) problem is described as "determining the set of roles to be activated in a single session for a particular set of permissions requested by the user" [4]. Further, the UAQ problem is better resolved by assigning a lower bound or an upper bound to the permissions being activated for a particular incoming request [5]. In [6, 7] the optimization objective for extra permissions are considered but failed to consider the missing permissions when they formulated UAQ as a joint optimization objective. Hence, [8] came forward with an optimization objective where the UAQ problem was divided into two subcategories known as: Core-UAQ problem and the constrained-UAQ problem. Core-UAQ introduces irreducibility, permission-cardinality and role-cardinality constraints. According to [8], a RBAC system optionally contains any Core-UAQ or Constrained-UAQ. Research is also carried out to solve the UAQ problem with constraints like dynamic mutually-exclusive roles (DMER), where users should not be activated with more than certain number of roles at a time [9]. Rapid growth of research in this field can be observed. The UAQ problem is now being considered as one of the prime issues regarding effective handling of incoming users' requests. We must ensure that our algorithm selects a set

of roles that covers the desired set of permissions requested by the user in RBAC.

When determining which set of roles should be activated in a session, one has to ensure that the desired permissions are covered by the roles and hence are available to the session. A user may need a set of permissions which are not included in a single role. In this case two or more roles need to be merged or clubbed to satisfy the user's request. However, such merging may not be haphazard as the following three reasons are listed. Firstly, every company or organization consists of lower level roles and higher level roles, where lower level roles are often a subset of higher level roles. Merging haphazardly may result in merging a lower level role with a high level role which equals to the higher level role itself. Secondly, in organizations a single user may have multiple roles and sessions. These roles might be divided based on the sessions called Dynamic Separation of Duty relations aka DSD relations [10]. This means there is a restrict on the number of roles that can be activated in a given user's session. As an example, an employee can request for a product and he can also approve the request. However, he should not be able to validate his request. He may authorize others' requests. Thirdly, while merging we also need to take care that no two or more roles of different users are combined because user may get unauthorized privileges which is against the principle of RBAC mechanism. Hence, in this paper, we suggest merging of roles of a single user which contain similar types of permissions in a given session, store them and do the same for the remaining users.

We introduce an efficient model to compute a new set of roles in addition to the existing sets of roles by merging the existing sets of roles. Here, the DSD relations and *principle of least privilege* (the user must be able to access only the roles that are necessary for its legitimate purpose) is also taken care of. For this purpose, we use unsupervised machine learning technique known as Agglomerative Hierarchical Clustering, where the merging is determined based on minimum Euclidean distance between the permissions of those roles. This algorithm efficiently computes the difference between the permissions and combines the roles with a minimum difference in their permissions.

Our Contributions:

1. The proposed model is developed using Agglomerative Hierarchical Clustering to solve the UAQ problem in RBAC systems.
2. The proposed model satisfies the constraints on dynamic separation of duties in RBAC system by generating required roles and matching the

incoming requests in cubic and linear time respectively, which is more practical and efficient compared to the state-of-the-art.

The rest of the paper is organized as follows: Section 2 discuss the related works. In Section 3, the proposed methodology is provided. In Section 4, the experimental results are presented and finally, Section 5 concludes the paper.

2 Related Work

An user's request consists of a set of permissions needed for their particular session. A particular role or set of roles, which contain these permissions, are chosen and assigned to this user. Various approaches have been utilized to choose the minimal, maximal or exact set of roles to satisfy the request. Some of them are discussed next. The first approach demonstrated to optimize the solution for UAQ can be found using the greedy approach to search for requested permissions from existing set of roles [4]. In [11], the UAQ problem is considered as a multi-set of DNA strands, and hence implements the DNA algorithm. Here the problem is divided into sub-cases like minimal match, maximal match and exact match. In [5], two methods are suggested to tackle the UAQ problem. First method uses a backtracking algorithm to find the best match to any incoming request. This algorithm proves efficient if the cost of the path to be traversed is more than the already traversed path. However, in worst case this algorithm takes exponential time. Second method reduces the UAQ problem into satisfiability problem (SAT) which works better than backtracking for minimal match case and maximal match case but not for exact match. In [12] the UAQ problem was transformed into SAT and solved using *sat4j* java library. Furthermore, in [8], focus was to decrease the number of permissions assigned to a user by implementing a static pruning technique. It also employs a preprocessing technique to minimize the amount of roles to be taken into consideration and uses depth-first search based algorithm to solve the UAQ problem. A new and innovative approach towards the UAQ problem is explained in [13] by assigning weights to every permission based on its role. The UAQ problem is converted to a corresponding chromosome and the binary evolution algorithm is used to solve it. In Table 1, we provide a comparison of the proposed approach with the existing algorithms.

All existing approaches given in the Table 1 depends on the number of times the underlying algorithm is used. For example, the algorithm is triggered every time for the acquisition of a permission set requested by the user

Table 1 Properties of different user authorization query algorithms

Property	UAQ in RBAC [5]	DNA [11]	Role-Permission Reassignment [12]	Greedy Approach [8]	Proposed Approach
Algorithm	a. Backtracking b. Reduction to SAT; solving SAT	DNA-based computing	Reduction to SAT; using sat4j to solve	Pruning and DFS search	Hierarchical Agglomerative Clustering
Use of Algorithm	Every time a request comes in	Every time a request comes in	Every time a request comes in	Every time a request comes in	a. Only when roles or users are modified/added/deleted b. Every time a request comes in
Time Complexity	$O(2^n)$	$O(2^n)$	$O(2^n)$	$O(b^{d/2})$ b=breadth d=depth	$O(n^3)$ for generating the roles and $O(n)$ for matching the incoming requests
Role mapping sub-cases	Exact, minimal and maximal matches	Exact, minimal and maximal matches	Not applicable	Exact, safe and available matches	Exact and minimal matches

and to generate the required roles. If the algorithm is exponential and there are multiple requests, then some request might have to wait to get executed. This decreases the time efficiency of the model. Additionally, role mapping sub-cases such as exact, minimal and maximal matches helps in providing the required roles. Hence in Table 1, we have compared algorithm, time complexity, use of algorithm and role mapping sub-cases with the existing models. Unlike the existing models we do not need to run our algorithm every time when a new request for acquisition of permissions comes in. Proposed algorithm needs to be executed only when the administrator adds new roles or users or modifies existing roles or DSD relations. To perform such operations, the proposed algorithm takes cubic time complexity of $O(n^3)$ and to match the incoming request algorithm takes linear time complexity of $O(n)$. Further, only exact and minimal matches are considered for the requested permissions because maximal match need not necessarily provide all the requested permissions and carries a risk of additional permissions being assigned to the user. The main objective of solving the UAQ problem for RBAC is acquisition of a permission set requested by the user by generating and assigning the necessary roles based on the session.

3 Methodology

In this section, we show the design and implementation of the proposed model to optimize the solution for UAQ in RBAC systems.

Supervised learning algorithms need a labelled dataset which tells the algorithm whether their predictions/answers are right or wrong. For implementing such algorithms to solve UAQ we would need to supply massive

amounts of data with labels (such as user requests across different sessions with the right set of roles as outputs) to train the algorithm. Unsupervised learning, however, does not require a labelled dataset, which means it does not need to go through the training phase. It can directly be tested. In short, it saves our time and effort to generate massive labelled datasets. Reinforcement learning is used in cases where datasets are not available. In UAQ, a set of roles are predefined with certain constraints by the company. Thus, unsupervised learning seems to be the best option for the UAQ problem in RBAC systems.

3.1 Design Architecture

Here, the design architecture of the proposed model is presented in Figure 1. The architecture depicts the control flow of the proposed model and the detailed description of the activities are given below:

1. A binary dataset containing user-role and role-permission assignments along with DSD relations are fed into the computational module. This is done by an admin when roles or users are created or modified.
2. An unsupervised machine learning technique, agglomerative hierarchical clustering is used to form clusters of similar roles for each user. If the number of users is n in the input, this algorithm runs n times.

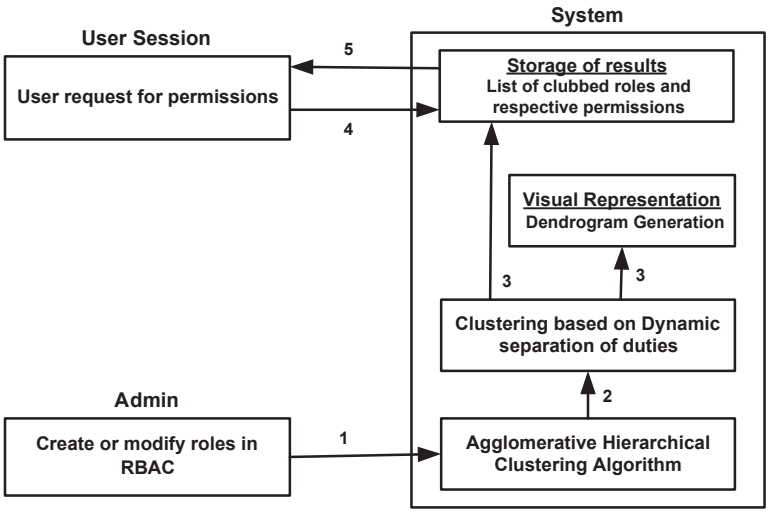

Figure 1 Optimized solution for UAQ using agglomerative hierarchical clustering.

3. All the roles clustered by the algorithm according to the user following DSD relations are stored in a data structure or can be directly written to a file. Thus, these clusters (i.e., optimized set of roles) are stored and the result file may be overwritten or updated with these clustered roles. Additionally, respective dendrogram is drawn which accurately represents the clusters for each user's roles.
4. A user wanting to gain access to particular permissions requests the system to obtain optimized set of roles. This request contains a list of permissions.
5. This request is run through various permission set corresponding to the clubbed roles of the particular user. The matching permission set is chosen and its respective role(s) is/are assigned.

3.2 Implementation

3.2.1 Definitions

Here we shall see some definitions which are used throughout this paper.

1. Nodes[ ]: A list data type where every row of Nodes[] list contains the resulting clusters formed by the Agglomerative hierarchical clustering.
2. ROLES = $\{R_1, R_2, \ldots, R_k\}$: Set of roles.
3. PERMS = $\{P_1, P_2, \ldots, P_n\}$: Set of permissions.
4. N_{R_u}: Total number of roles of a particular user u.
5. Dynamic separation of duty (DSD) $\subseteq (2^{ROLES} \times N)$ is collection of pairs (rs, n) in DSD, where rs is a set of roles, n is a natural number with maximum number of roles in rs, i.e., $1 \leq n \leq |rs|$ for $rs \subseteq$ ROLES. Here the DSD relation implies that no user is activated with n or more roles from the role set rs in each $(rs, n) \in$ DSD.

3.2.2 Algorithms

Initially, the Algorithm 1 reads the dataset containing role assignments to users and permissions, performs clustering, creates dendrogram and stores the result. The $generateDSDRelations$ generates the DSD relations by taking care that no randomly generated role set has N or more than N number of roles. The $AgglomerativeClustering$ method returns a set of mentioned numbered clusters. Euclidean affinity refers to how the distance is calculated and applying ward's method for linkage. Suppose there are 30 roles in the dataset for a single user, then setting $i = 30$ will return 30 numbered points i.e., 30 roles. In the next iteration i is decremented by one so it returns 29 numbered clusters. Which means that 2 nearest roles have

Algorithm 1: Agglomerative Hierarchical Clustering based solution for UAQ.

Require: Binary dataset
Ensure: Clustered role and permission set
$read(dataset.csv)$
Nodes[] $\leftarrow \phi$
$generateDSDRelations(N_{R_u}, N)$
for $i \in [N_{R_u}, 1]$ **do**
$//decrementing\ loop$
Nodes[] $\leftarrow$ Nodes + $AgglomerativeClustering(clusters = i,$
$affinity = euclidean, linkage = ward)$
end for
ClubbedRolesList $\leftarrow generateClubbedRolesList(Nodes[])$
$generateDendrogram(ClubbedRolesList)$
StorageOfResults $\leftarrow createFinalList(ClubbedRolesList)$

Algorithm 2: Searching the roles for the incoming permissions.

Require: Permission set and StorageOfResults.
Ensure: Assignment of optimized set of roles to the user.
OptimizedRoles $\leftarrow$ linearSearch($\{P_i, \ldots, \}, StorageOfResults$)
$Assign(OptimizedRoles)$

been clubbed into a cluster, where nearest here means that these roles have very similar permissions. Thus, this is how we get our set of clustered roles. Specific libraries for this function are available in various languages. The $generateClubbedRolesList$ method analyses every row in the Nodes[] list. Finds the optimized roles by merging into a cluster. If they belong to the same DSD role set, it adds them to $ClubbedRolesList$. Else, it splits them according to the DSD role sets and adds them to the $ClubbedRolesList$. The $generateDendrogram$ method generates the pictorial representation of how the hierarchical clustering of roles are supposed to be clubbed for each user. The $createFinalList$ method creates merged permissions according to the merged roles from $AgglomerativeClustering$ and stores them according to the user and duplication is also removed using this method. The time complexity of hierarchical algorithm is $O(c * n^2)$, where n represents the number of elements to be clustered and c represents the number of generated clusters. In the proposed approach, since the value of c is equal to n, the time complexity is $O(n^3)$. The Algorithm 2 takes a set of permissions as an input from an user, searches for requested permissions from his results file and assigns a set of optimized roles to the user based on the session.

4 Experimental Results

In this section, we show some experimental results obtained by applying the proposed algorithm to real world datasets and compare its performance with existing algorithms using the synthetic dataset. All the test cases have been carried out on HP-Probook $440\ g4$ with Intel Core $i5(7thGen)$ Processor and 4 GB RAM.

4.1 Real World Datasets

The proposed algorithm has been applied on real world datasets available online at HP labs [14]. The number of optimized set of roles and its corresponding time taken for the nine real world datasets are summarized in Table 2. The first column represents the name of the binary dataset, and its corresponding characteristics are found in [15]. The second column represents the number of roles generated using simple role mining algorithm. The third column represents the number of clusters or optimized set of roles generated using the proposed model. Finally, the fourth column represents the time taken to generate the optimized set of roles. But for simplicity and understanding of the algorithm, here, we consider only one user assigned to different combinations of 8 permissions through 16 roles as shown in Table 3. Observing Table 3, we understand that roles R_9 and R_{10} are most similar to each other because they share many common permissions. Therefore before Agglomerative Clustering, $Nodes \leftarrow []$ and after Agglomerative Clustering the nodes are given in Figure 2. The first row in the list signifies that there are 16 clusters made from 16 roles. Next iteration of the for loop returns 15 roles where roles R_1 and R_8 have been clubbed as represented by $'0'$

Table 2 Summary of the results obtained using real world datasets

Dataset	Generated Roles Using Simple Role Mining Algorithm [15]	Number of Optimized Set of Roles	Time Taken (seconds)
am. large	430	429	102.6
am. small	225	224	58.4
apj	475	474	3.8
emea	34	33	4
healthcare	16	15	0.83
domino	20	19	0.41
customer	1154	1153	25.23
firewall1	71	70	17.75
firewall2	10	9	20.24

Table 3 Role-Permission assignments for a user in RBAC

Roles	Permissions							
	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
R_0	1	1	1	1	0	0	0	0
R_1	1	0	1	0	1	0	0	0
R_2	0	1	0	0	0	1	1	0
R_3	0	0	1	1	1	0	1	0
R_4	1	0	1	0	0	1	1	0
R_5	0	0	1	0	1	0	1	1
R_6	1	1	0	0	0	0	1	1
R_7	0	1	1	1	0	1	0	0
R_8	1	0	1	1	1	0	0	0
R_9	0	0	0	1	1	1	0	0
R_{10}	0	1	0	1	1	1	0	0
R_{11}	0	0	0	1	1	0	1	1
R_{12}	1	1	0	0	0	1	0	1
R_{13}	0	1	1	0	1	1	0	0
R_{14}	1	1	0	0	0	1	1	0
R_{15}	0	0	1	0	0	1	0	1

Nodes ⟵ [[12 15 14 11 8 10 9 7 13 6 5 4 2 3 1 0]
[12 0 14 11 8 10 9 7 0 13 5 4 6 3 1 2]
[12 1 0 11 8 10 9 7 1 13 5 4 6 3 0 2]
[12 1 2 11 8 10 9 7 1 0 0 4 6 3 2 5]
[0 3 2 11 8 10 9 0 3 1 1 4 6 7 2 5]
[1 3 2 0 8 0 9 1 3 4 4 10 6 7 2 5]
[1 3 2 0 8 0 9 1 3 4 4 0 6 7 2 5]
[1 3 6 2 8 2 0 1 3 4 4 2 0 7 6 5]
[1 3 0 6 0 6 2 1 3 4 4 6 2 7 0 5]
[4 3 1 6 1 6 2 4 3 0 0 6 2 0 1 5]
[4 3 0 2 0 2 0 4 3 1 1 2 0 1 0 5]
[0 3 1 2 1 2 1 0 3 4 4 2 1 4 1 0]
[0 3 1 2 1 2 1 0 3 0 0 2 1 0 1 0]
[2 0 1 0 1 0 1 2 0 2 2 0 1 2 1 2]
[0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 0]]

Figure 2 Resulting clusters formed by the agglomerative hierarchical clustering.

at $Nodes[1][1]$ and $Nodes[1][8]$. Similarly, the next iteration shows that roles R_2 and R_{14} have been clubbed represented by '0' at $Nodes[2][2]$ and $Nodes[2][14]$ as well as and represented by '1' at $Nodes[2][1]$ and $Nodes[2][8]$. This means roles R_2 and R_{14} are in one cluster represented by '0' and roles R_1 and R_8 are in another cluster represented by '1'. A diagram of a dendrogram makes it more clear and easy to understand the

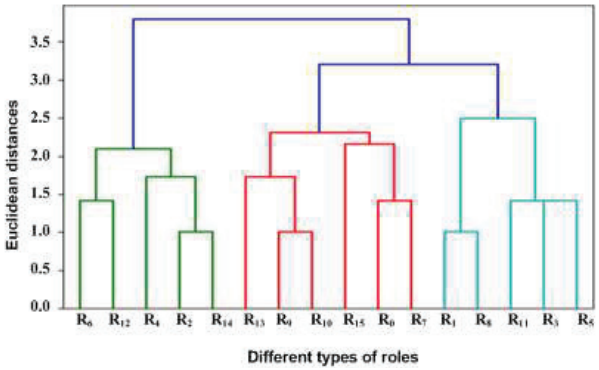

Figure 3 A Dendrogram shows the relationship between roles from Agglomerative hierarchical clustering in RBAC.

hierarchical relationship between roles as shown in Figure 3. The Figure 3 illustrates the arrangement of the clustering for roles based on the increasing order of Euclidean distance.

Since the real dataset do not have DSD relations, we created it using the *generateDSDRelation* function for the roles specified in Table 3 with the value of $N = 9$. The generated role sets are $rs_1 = \{R_1, R_5, R_2, R_0, R_3, R_4, R_5, R_6\}$ and $rs_2 = \{R_{10}, R_9, R_{15}, R_{13}, R_{11}, R_{14}, R_8, R_{12}\}$. This means role sets rs_1 and rs_2 cannot be clubbed together or activated simultaneously. In the dendrogram, i.e., in Figure 3, we observe that roles R_2 and R_{14} have been clubbed (this is done in iteration 3 as shown in Figure 2 which is represented by '0' in the nodes list at positions $Nodes[2][2]$ and $Nodes[2][14]$). But as we know, R_2 and R_{14} belong to different DSD role sets and are not supposed to be activated simultaneously. Thus, the algorithm discard such merging of roles across different DSD role sets in the *generateClubbedRolesList* and the corresponding dendrogram is shown in the Figure 4. We also observe that roles R_9 and R_{10} have been clubbed in the second iteration of $AgglomerativeClustering$ function, and also shown in the Figure 4, but not included in the roles and permissions list below. The reason for that is roles R_9 and R_{10} when combined give us the same set of permissions as individual role R_{10} itself. Role R_{10} has already been added to the list before clubbing roles (R_9, R_{10}). Hence, to remove duplication, clubbed role (R_9, R_{10}) has not been added to the final list and such role duplication has been taken in the $createFinalList$ function. The final lists of clusters (i.e., optimized set of roles) and their permissions is given in Figure 4.

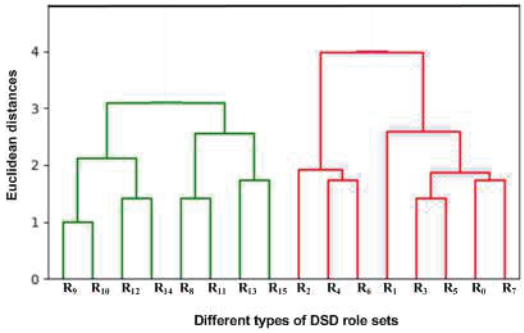

Figure 4 A dendrogram from agglomerative hierarchical clustering by enforcing dynamic separation of duty in RBAC.

Table 4 The optimized set of roles without role duplication and its corresponding permissions

Clusters (Optimized Set of Roles)	Permissions							
	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
R_3, R_5	0	0	1	1	1	0	1	1
R_{12}, R_{14}	1	1	0	0	0	1	1	1
R_8, R_{11}	1	0	1	1	1	0	1	1
R_{13}, R_{15}	0	1	1	0	1	1	0	1
R_4, R_6	1	1	1	0	0	1	1	1
R_0, R_7	1	1	1	1	0	1	0	0
R_2, R_4, R_6	1	1	1	0	0	1	1	1
R_0, R_3, R_5, R_7	1	1	1	1	1	1	1	1
R_{10}, R_{12}, R_{14}	1	1	0	1	1	1	1	1
R_1, R_3, R_5	1	1	1	1	1	0	1	1
R_8, R_{11}, R_{13}	1	1	1	1	1	1	1	1

4.2 Synthetic Dataset

In this section, we will compare our model with the research work mentioned in the comparison Table 1 using the synthetic dataset generated by the synthetic data generator as suggested in [16]. Synthetic data generator takes as input the number of users (#U), number of roles (#R), number of permissions (#P), maximum number of roles assigned for a user and maximum number of permissions for a role as shown in Figure 5. Further, as shown in Figure 5, the synthetic dataset is generated on varying maximum number of roles assigned for a user, while the other input parameters are constant. Since the synthetic dataset is generated randomly, here we report the average results by running the heuristics 5 times for each dataset.

Table 5 Users with a varying maximum number of assigned roles

#U	#R	#P	#Maximum Roles For a User	#Maximum Permissions For a User
5	2000	4000	500	5
5	2000	4000	1000	5
5	2000	4000	1500	5
5	2000	4000	1800	5

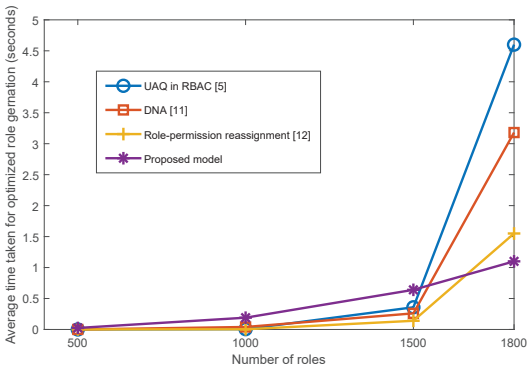

Figure 5 Comparison of the time taken by various UAQ algorithms for optimized role generation.

Figure 5 presents a plot containing the average time taken for optimized role generation on the y-axis against the varying number of roles used in the algorithms on the x-axis. The average time taken by the proposed model varies from 24 milli seconds to 1.1 seconds over number of roles varying from 500 to 1800. Further, the average time taken by the proposed algorithm performs effectively for large number of roles when compare to the existing work for the same operation with exponential time.

In [12], the method for role generation is not included so we could not compare our clustering algorithm with it. However, in [12], it uses greedy approach to search and match for the incoming requests. Figure 6 presents a graph containing the average time taken to search and match the requested permissions on the y-axis against the varying number of roles used in the algorithms on the x-axis. The average time taken by the proposed model varies from 50 milli seconds to 180 milli seconds over number of roles varying from 500 to 2000. Therefore the proposed model depicts linear growth where as the greedy approach [12] depicts linearithmic growth. Thus, the proposed model is most suitable for applications where large number of

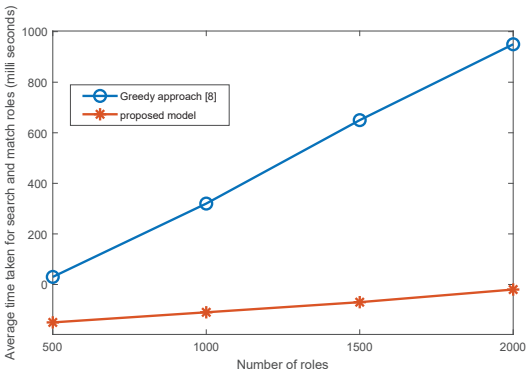

Figure 6 Comparison of the time taken for search and match the incoming request.

roles are assigned for users. Additionally, in the proposed model the maximal match (which do not provide the requested permissions) do not occur because in hierarchical clustering the last generated cluster contain all the roles that can be assigned to the user in a given session (refer Figures 3 and 4).

In [9], the authors consider parametric in some relevant dimensions of UAQ problem such as number of roles, number of DMER constraints, number of requested permissions and maximum number of roles that can be activated. However, these dimensions of UAQ problem do not focus much on the RBAC's core principle of least privilege, unlike the proposed model. Further, it is also observed in [4, 5, 9] that solving UAQ problem using search based techniques take exponential time when compared to the proposed model with cubic time.

5 Conclusion

This paper demonstrates that the user authorization query problem can be efficiently solved using an unsupervised machine learning technique i.e. Agglomerative Hierarchical Clustering with computational efficiency in RBAC systems. The model described here computes clusters (i.e., optimized set of roles) by combining existing roles of a user which are similar and avoids the duplication in the roles. Based on the requested permissions by the user in a given session, the algorithm finds the exact or minimal match in linear time. Unlike the existing proposals the proposed approach generates clusters only if there is change in the role assignments to users and permissions. Additionally, it guarantees the requested permissions following dynamic separation of duties in RBAC.

The experiment was conducted on real world datasets available on HP labs. The analysis was performed by considering certain role-permission assignments of a user and a diagram of dendrogram makes easy to visualize the clustering of roles. The performance evaluation of the proposed algorithm compared to the existing algorithms are done by generating synthetic dataset. With a complexity of $O(n^3)$ this approach proves to be one of the fastest and promising models in the state-of-the-art. The proposed model does not consider weighted roles which is considered as our future work.

References

[1] F. David and K. Richard. Role-based access controls. In *Proceedings of 15th NIST-NCSC National Computer Security Conference*, volume 563. Baltimore, Maryland: NIST-NCSC, 1992.

[2] R. Sandhu, D. Ferraiolo, and R. Kuhn. The nist model for role-based access control: towards a unified standard. In *ACM workshop on Role-based access control*, volume 10, 2000.

[3] K. Rajesh Rao, A. Nayak, I.G. Ray, Y. Rahulamathavan, and M. Rajarajan. Role recommender-rbac: Optimizing user-role assignments in rbac. *Computer Communications*, 166:140–153, 2021.

[4] Y. Zhang and J.B.D. Joshi. Uaq: a framework for user authorization query processing in rbac extended with hybrid hierarchy and constraints. In *Proceedings of the 13th ACM symposium on Access control models and technologies*, pages 83–92, 2008.

[5] G.T. Wickramaarachchi, W.H. Qardaji, and N. Li. An efficient framework for user authorization queries in rbac systems. In *Proceedings of the 14th ACM symposium on Access control models and technologies*, pages 23–32, 2009.

[6] N. Mousavi and M.V. Tripunitara. Mitigating the intractability of the user authorization query problem in role-based access control (rbac). In *International Conference on Network and System Security*, pages 516–529, 2012.

[7] N. Mousavi. Algorithmic Problems in Access Control. Ph.d. dissertation, University of Waterloo, Canada, 2014.

[8] J. Lu, J.B.D. Joshi, L. Jin, and Y. Liu. Towards complexity analysis of user authorization query problem in rbac. *Computers & Security*, 48:116–130, 2015.

[9] A. Armando, G. Gazzarata, and F. Turkmen. Benchmarking uaq solvers. In *Proceedings of the 25th ACM Symposium on Access Control Models and Technologies*, pages 145–152, 2020.
[10] ANSI INCITS. Incits 359-2004, american national standard for information technology, role based access control. *American National Standards Institute*, 2004.
[11] Z. Tang, R. Guan, and K. Li. User authorization queries in rbac systems based on dna computation. In *2010 IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA)*, pages 174–179, 2010.
[12] J. Lu, Y. Xin, Z. Zhang, H. Peng, and J. Han. Supporting user authorization queries in rbac systems by role–permission reassignment. *Future Generation Computer Systems*, 88:707–717, 2018.
[13] J. Lu, Z. Wang, D.Xu, C. Tang, and J. Han. Towards an efficient approximate solution for the weighted user authorization query problem. *IEICE TRANSACTIONS on Information and Systems*, 100(8):1762–1769, 2017.
[14] R Schreiber. Datasets used for role mining experiments.
[15] C. Blundo and S. Cimato. A simple role mining algorithm. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, pages 1958–1962, 2010.
[16] J. Vaidya, V. Atluri, and J. Warner. Roleminer: mining roles using subset enumeration. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 144–153, 2006.

Biographies

K. Rajesh Rao received his B.E. degree in Computer Science and Engineering and an M.Tech. degree in Computer Science and Information Security. His Ph.D. degree is in the area of Cloud Information Security from Manipal

Academy of Higher Education (MAHE), Manipal, India. Currently, he is an Assistant Professor-Senior at Manipal Institute of Technology, MAHE, and is also associated with City, University of London as a Researcher in the area of cyber security. His research interests include, but are not limited to security analytics, access control models, cloud security, internet of things, and soft computing.

Aditya Kolpe received his BTech in Information Technology from Manipal Institute of Technology, Manipal and currently working as Associate Software Engineer at Oracle. His research interests include cyber security, data science and machine learning.

Tribikram Pradhan received his Ph.D. from Indian Institute of Technology (BHU), Varanasi in 2020, where he contributed to the development of a Multi-objective academic recommender system to provide recommendations for papers, citations, collaborators, reviewers, and academic venues. He also proposed a model for automatic meta-review generation considering individual reviews of a given research paper. Prior to joining IIT (BHU), he also worked as an assistant professor in the Department of Information and

Communication Technology, Manipal Institute of Technology, Manipal. His research interests include information retrieval, recommender systems, text mining, social network analysis and natural language processing.

Bruno Bogaz Zarpelão received his BSc degree in computer Science from State University of Londrina, Brazil, and the PhD degree in Electrical Engineering from University of Campinas, Brazil. He is currently an Assistant Professor at the Computer Science Department of the State University of Londrina (UEL), which he joined in 2012. From March 2018 to February 2019, he was a visiting postdoctoral researcher with City, University of London. His research interests include security analytics, machine learning applied to cyber security, and internet of things.