International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-9 Issue-3, January 2020

A Novel Perspective for Organizing the Conflicting Security Tasks in a Dynamically Distributed Systems Environment

GSN Murthy, T. Veerraju, T. Satya Kumari, V. Anantha Lakshmi

Abstract: ***Distributed systems environment comprises set of Hosts, workstations, terminals, Different Servers, bridges, Gateways, printers and so on. The sharing of resources and information in this distributed systems and networking environment increases day-by-day. This is more and more prone to attacks and hazards. Now-a-days, as threats become more widespread, attackers get smarter and the infrastructure required securing. Hence, in large systems, it is essential to distribute the management tasks by defining the security policies by the systems manager for controlling, monitoring and organizing employee's access. In some cases, it is possible to employee's tasks may conflict or overlap. To avoid this, we present a Novel approach for organizing Conflicting Security tasks that has been specifically designed with consistency. It organizes tasks into a tree shaped structure to maintain significant representation.***

Keywords: Conflicting, Privacy, Security Policy, Task.

I. INTRODUCTION

The most important feature of any information system in the distributed environment is the guarding of information against illegal disclosure. At the same time, authenticity and authorization are also the important issues in this environment along with the confidentiality. Such protection ranges from simple authentication to the most complex authorization required for susceptible information. Networks and distributed systems can be very large and complex which requiring the partitioning of management responsibility according to functions. But the partitioning may also reflect organizational, geographical or even network layer boundaries.

Partitioning of duties is a good practice for avoidance of fraud and by specifying the policies in a systematic manner by partitioning the tasks and its related privileges.

Revised Manuscript Received on January 30, 2020.
***** Correspondence Author
Dr.G.S.N Murthy*, Professor, Dept of CSE, Aditya College of Engineering, Surampalem, India, Email: murthygsnm@yahoo.com
T.Veerraju, Associate Professor, Dept of CSE, Aditya College of Engineering , Surampalem, India.
T.Satya Kumari, Assistant Professor Dept of CSE, Aditya College of Engineering, Surampalem, India.
V.Anantha Lakshmi, Associate Professor, Dept of CSE, Aditya College of Engineering, Surampalem, India.

© The Authors. Published by Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP). This is an open access article under the CC-BY-NC-ND license http://creativecommons.org/licenses/by-nc-nd/4.0/

These policies consist of a chain of restrictions which are linked with a set of tasks which, may be allocated to one or more employees according to their job. Permissions are associated with tasks and users/Employees (Here after, we use as an employee) are made members of those tasks for organizing Conflicting Security tasks to make sure that they are consistent. On the other hand, in speedy altering systems, an employee would likely have more than one task, and some of these tasks may overlap.

A significant effort has been made to properly defining policies in information systems in the distributed environment to avoid conflicts. In fact, three levels of policy specifications were identified [1]. The first one is the Top-level abstract policies, which can be objectives of the business, different agreements, or trust relationships. These policies are not enforceable and their realization involves refining them into one of the other two policy levels. The second level is the Middle-level policies or business-level policies, which are specified by the system manager and related to specific objects and finally the Bottom-level policies such as security mechanism patterns or machine patterns that are related to s/w & h/w. In this work, we focus on the Specification policies and discuss the concepts used to express these policies. After that we present a novel approach that is designed to organize the conflicting policies Tasks. In addition to that algorithms are also given for assigning a new task and deleting an existing task.

Specification policies determines which an employee may access specific information. It can be able by defining a series of conditions (constraints) by the system manager for controlling monitoring and organizing the access [2]. In common, a rule is resolute based on the sensitivity of the information. If it is sensitive, a policy should be developed to sustain tight control over accessing that information. For example, within a well-established industry/Organization, the financial profits and liabilities has to known by the top-Level officials only. It is to be considered as the most sensitive and any other employee should be restricted to access/enter into the resources. Like that, In a Military organizations, the information related to the strength of the country in terms of weapons, rifles, Machine guns, canons, rockets, grenades, Mortars, Pistols, Howitzers and the Armed forces has to known by the higher officials only. This information is more and more sensitive and the access is restricted to the common people and other Bottom-level officials like soldiers and etc.

In similar, the Top rated government secrets which are most sensitive can be accessed by only the Higher-Level bureaucrats and secretaries to the Government.

This information should be restricted to the Lower-Level officials and disgruntled employees working within the Government organization. In fact, safety considerations range from simple policies to the composite and complicate policies to protect the sensitive information.

Supporting such policies is based on a combining of three essential elements Employees, Objects and Tasks. Employees are to which authorizations are granted. Employee/employees can be an individual or a group of persons within the organization. Data to be protected can be any part of the stored information. Lastly, Tasks, which are named collections of authorizations or privileges granted to employees to perform certain job functions. For example, let us assume that an organization in which Tasks are created based on the job functions of the employees. Tasks are subsequently having a set of Constraints based on the requirements of the jobs/Functionalities. Employees are assigned appropriate tasks based on their eligibility like Age, Designation, qualification, service in the particular organization, and their trustworthiness etc.,.

In majority of the cases, an employee needs to perform more than one task, with overlapping permissions. These tasks may be planned into a hierarchy to hold a more expressive illustration. As result the any conflicting authorizations may be resolved. An employee with more than one task can have rights to access the tasks authorized and restricted by the other tasks. In such type of cases, the employee is supposed to be given the highest permissions amongst his/her tasks by discovery the highest non-conflicting tasks. To deal with such matters, our approach organizes the tasks into a hierarchical structure (i.e., tree shaped structure) using an open source library called tree.hh used by [3]. Then the inheritance is resolved in the perspective of overlapping tasks. Finally, the non-conflicting authorizations among the assigned tasks are extracted. The clear explanation about this approach is discussed in section 4.

The remaining part of this paper is organized as follows: Section 2 describes the Background information related to this approach and motivation behind this work. Our methodology is discussed in Section 3. The proposed approach itself is then presented in Section 4 along with its result analysis in Section 5. Final conclusion is given in Section 6.

II. BACKGROUND INFORMATION & MOTIVATION

While starting with the explanation about our novel approach, we provide a concise general idea of the essential terms and the organizational structure related to the policies.

1.1 Employees, Objects and Tasks: In certainty, policies consist of tasks, data/objects, and employees. Tasks are primary formed with precise constraints or authorization on particular objects (data) based on the job functions. Employees would be an employee or even a program permitted to do particular operations based on their assigned duties. Such elements are particularly useful for common operations such as adding/deleting an employee, or assigning/altering employee's tasks. Fig 1 portrays the associations. The merging of access control used for several employees into a unique task entry allows easier for the organization of the approach and much efficient verifiability of secured policy. In an outsized distributed systems environment, task hierarchy and the necessity for customized privileges makes supervision potentially clumsy. The organizing of an individual's constraints becomes simpler in that constraints do not have to be directly assigned on an Employee-by-Employee basis. Some other details about tasks hierarchy is explained later.

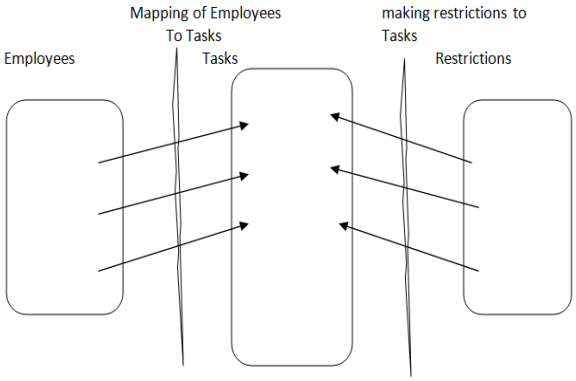

Fig 1: Employees, Tasks, Restrictions relationships

2.2 Tasks Hierarchy

Task: A task is an activity or a piece of work assigned to a specific one without conflict. Tasks can be linked together to create dependencies.

Tasks may have ordered into a hierarchy which, characterize as a quasi ordering, symbolize as $\preccurlyeq$ [4]. These hierarchies possess a more open imagination especially in the case of existing overlapping permissions.

Def 1: No employee in the environment should be allocate to two conflicting tasks. Conflicting tasks can not have the common employees. It can be expressed as

(OT (CT) $\in$ *Tasks (OT (E))* == > OT (CT) $\cap$ tasks (OT (E)) = Ø.

Where

OT=one task,
CT= Conflicting tasks,
E= All the Employees in the Environment.

Ex: Permission to make soft/hard copy of the work for personal use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage. To copy otherwise, to post on servers, or to use this lists requires a prior specific consent and/or a fee.

Like that, we can define more about this, as a given a task T, let t_1, $t_{2...}$ $t_n \in$ T be individual Tasks. If t_1 is at the top level and t_2 is at the next level in the hierarchy ordering and additionally, that t_1 is a child of t_2 and t_2 is a parent of t_1, then t_1 becomes heir to all constraints that are assigned to t_2 also, and that all employees who are mapped to t_1 are affected by the t_2 constraints. Similarly if t_2 precedes t_3 in the hierarchy ordering the same what mapped to t_1 and t_2 are happened to t_2 and t_3 which is articulated in Def 2.

Def 2: A task t_1 in a task hierarchy T heir all the constraints of the remaining tasks in T= (t_2….t_n), which implies, t_1 heir all the constraints of the tasks reachable from t_1, to the head task T.

Task hierarchy is illustrated in Fig 2 as an example, where any task inherits all the constraints that are assigned to its parents up to the Head. For example, suppose that pathological information of a patient in a patient table which consists of the attributes (e.g., Name, Age, city, Test Result) of an orthopedic division can be accessed by the corresponding Doctor and nurse staff only. It should not be accessed by the doctors and the nurse staff of other specializations like Nephrology. Consequently, the doctors and the nurse staff of an orthopedic division are restricted from accessing the patient table attributes of the specified table. This is called Security Object and defined in Def 3.

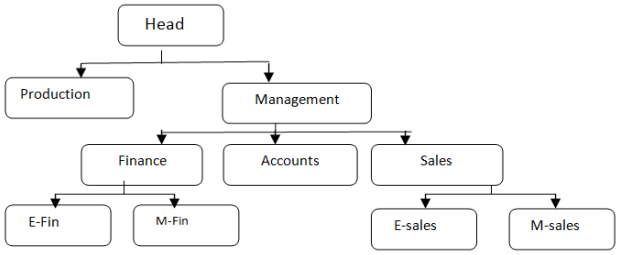

Fig 2: Tasks Hierarchy

Def 3: A Security Object O for tasks T = {{v}: {v}:= values of tables T_1 restricted to tasks T, where {v}:= all T's attributes values}.

Security Policies are taken into the consideration for the survey of the literature. For example, many languages are designed or extended for expressing policies such as the XML concepts based languages [5] and the logic programming based languages [6]. The extensible Access Control Markup Language (XACML) is the most pertinent proposed Language [7] is appropriate to express the requirements connected to permission for web-based systems.

Some Investigators measured the policies formulation [8]. For instance, the Unified Modeling Language (UML) to articulated dependence policies with predicate terms whose grammar is expressed in UML. Correspondingly, a Trust organizing structure that supports rule existence cycle management using UML diagrams is introduced [9]. Specific Policy design in web-based applications has been proposed [10]. An Abstract hierarchy Model for Managing Overlapping Security Roles model is designed by Altamimi, 2017. In this model, Description about the objects and restrictions to authorize by specifying security policies was designed. Other frameworks investigate the association of security policies with client side code with protection provided by the interception and analysis of database queries [11].

A framework for policy driven management for distributed systems was designed [12]. A family of reference models for Role-based access control was introduced [13]. A role based framework for distributed systems management also investigated [14]. For the most part, however, none of these works are designed to manage or consider the conflicting tasks but the policy designs itself. Using the RSL 99 Language, Role-based separation of duty (SOD) constraints technique is introduced [15] for prevention of fraud and errors and is practiced long before the existence of the computers. This SOD technique is unable to address the current demands. Because, the present system requires a dynamically changing environment, which are growing day by day to meet the current needs where the nodes are dynamically added and deleted.

III. METHODOLOGY

A novel approach for organizing the conflicting security tasks are required in dynamically changing distributed systems environment. It is designed to permit policy designers to identify the security constructs. In this work, we provide a simple algorithm also to rebuild the tree shaped structure after assigning a new task to the employee and after deleting an existing task from the totally available set of tasks. In a large Distributed systems environment, the organizing of the tasks is always dynamic and should not be static. The reason behind the dynamic nature is because of it is required to add a new task or need to delete an existing Task. In some cases, it is required to alter an existing task and needs to be assigned to some other due to the administrative constraints. With this objective, we propose a well-suited approach which is a tree-shaped nature. Tasks are organized into a tree-structured to support better illustration. Then the highest permissions amongst the assigned Tasks are removed by finding the Top-Level non-overlapping tasks. To resolve these problems, this approach relies on an open source library called tree.hh to systematize the tasks into a tree shaped structure, then the inheritance is resolved in the context of conflicting tasks and finally the Top-Level non-overlapping permissions are created. A more comprehensive conversation for our approach is provided in the later part of this section.

IV. A NOVEL APPROACH FOR ORGANIZING THE CONFLICTING SECURITY TASKS

In a quickly altering larger networking and distributed systems environment, it is necessary to be assign an employee for more than one task according to his/her jobs. Each task would have dissimilar permissions to access specific sensitive data, this should not be conflict. To avoid the conflicting of the tasks, most of the existing Distributed system environments apply a "constraint based priority" principle. The problem with this principle is if it is not well planned, resulting to lead complicated restrictions on accessible data. Assuming, if an employee is a member of several tasks with varied permissions, the employee will be controlled according to the permissions of the least prioritized task, results of restricting the employee from access to data even if she is permitted to access them by using other task(s).

For example, suppose the manager of a Distributed systems environment is included in the tasks shown in Figure 2 and assigns an employee Sreepriya to the task of manager, which has a complete access to the entire data relevant to finance, accounts and sales. Later, due to some administration constraints, if Sreepriya is assigned a task of sales, which restricts her from accessing Finance and accounting information.

Here in this case, the tasks assigned to the Sreepriya are conflicted. Sreepriya is restricted from accessing the Finance and accounts information because of she was assigned the task of sales.

However, she can access the Finance and accounts information because of she is already having the task of manager, who is the overall in charge of the tasks of Finance, accounts and sales.

To deal with this problem, Sreepriya should be given the top-Level permissions amongst her tasks by finding her top-Level non-overlapping tasks. To perform this, tasks should be organized in a tree shaped structure in order to pick-up the Top-Level task. In this structure a task -hierarchy restrictions will be inherited that a task of the below one take over all restrictions that are assigned to its top most hierarchy. Subsequently, all the employees who are mapped to this task are affected by the tasks restrictions plus all the inherited restrictions are expressed in Def 4.

Def 4: A User G who assigned to tasks $T_n = (t_1 \dots t_n)$ affected by all constraints of T_n PLUS all constraints of T_n with the hierarchies $H_n = (h_1 \dots h_n)$, where h_1 is the hierarchy for t_1 and $\dots h_n$ is the hierarchy for t_n, and h_x consists of $t_j, \dots, t_n$, where $t_1 \preccurlyeq t_2$ and $t_2 \preccurlyeq t_x \preccurlyeq t_n$ for all tasks $t_x \in T$. We say that G restricts by all constraints of T_n PLUS all constraints reachable from t_1 to the Head of T. Now, we discuss the implementation of this hierarchy along with a suitable example to demonstrate both the authorization and sensitiveness of our novel approach.

V. RESULT ANALYSIS

In our approach, the open source tree.hh library is used to implement the tree shaped hierarchy. It offer various types of iterators such as breadth first and depth first, to traverse the tree nodes where its access methods are compatible. DFS is recursive in nature, which introduces an overhead, and also might cause our code to hit the stack size limit for larger nodes. In our approach, we assumed a breadth first search strategy is the best fit. Because of its simplicity and works in a single stage. The algorithm traverses from the Head task and then explores the search in the level by level manner i.e., as close as possible from the Head task. We have to visit each node exactly once, since we do not traverse the same edge more than once. As such, the time complexity is O (n) where n is the number of tasks, which is very small.

Fig 3 illustrates the Task Tree associated with the tasks hierarchy depicted in Fig 2. Numbers with nodes represent the task ID. In the tasks tree, every node is connected to an arbitrary number of child nodes/Tasks. At the top of the tree, there may also exist, a set of tasks which are characterized by that do not have any parents. Nodes at the same level are called “siblings”' and are not overlapping. So, if an employee is assigned to sibling tasks, the employee's permissions will be the union of all her tasks restrictions. However, nodes at different levels may indeed conflict. Each node may inherit its parent's restrictions if any exist.

To improve the search performance, the employee’s task(s), along with their restrictions, are stored in a relational database. The Top-Level or most privileged tasks amongst the employee’s tasks are also stored in that. consequently, instead of re-executing the search process each time the employees request an access to the stored information, her Top most tasks are retrieved from the database and cached in memory for further accessing. Fig 4 illustrates an example for employees tasks stored in the database.

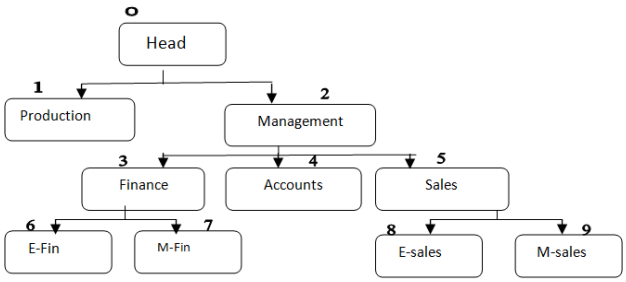

Fig 3: Tasks Hierarchy represented in a Tree based Structure

Fig 3 illustrates the Task Tree associated with the tasks hierarchy depicted in Fig 2. Numbers with nodes represent the task ID. In the tasks tree, every node is connected to an arbitrary number of child nodes/Tasks. At the top of the tree, there may also exist, a set of tasks which are characterized by that do not have any parents. Nodes at the same level are called “siblings”' and are not overlapping. So, if an employee is assigned to sibling tasks, the employee's permissions will be the union of all her tasks restrictions. However, nodes at different levels may indeed conflict. Each node may inherit its parent's restrictions if any exist.

To improve the search performance, the employee’s task(s), along with their restrictions, are stored in a relational database. The Top-Level or most privileged tasks amongst the employee’s tasks are also stored in that. consequently, instead of re-executing the search process each time the employees request an access to the stored information, her Top most tasks are retrieved from the database and cached in memory for further accessing. Fig 4 illustrates an example for employees tasks stored in the database.

Employees Tasks Table

Emp id	Emp Name	Task id	Task Name
1	Srivarshini	1	Production
1	Srivarshini	3	Finance
1	Srivarshini	5	sales
1	Srivarshini	6	e-sales
1	Srivarshini	8	e-Fin

Highest Tasks Table

Emp id	Task id	Task Name
1	1	production
1	3	Finance
1	5	sales

Fig 4: Employees Tasks in the policy Database

It is inevitable that in a dynamically changing environment, the employee's tasks can be altered or modified over the period of time based on their requirements and needs (e.g., Assign, Create, Add, Withdraw and Delete tasks). For example, assume that the employee Sreevarshini is assigned to the following tasks: sales, E-sales, Finance, E-Fin and production. Sreevarshini's restrictions will be defined by the union of these tasks. Note, E-sales is a child of the sales task, E-Fin is a child of the Finance task and because the employee utilizes the top most task. Hence, E-sales, E-Fin are not listed in the top most tasks table. As a result, its restrictions will be ignored.

Now, suppose the employee Sreevarshini becomes a manager. The tasks sales, E-sales, Finance, and E-Fin will not be listed in the top most tasks table,

and their restrictions will be disregarded in the security checking process because they are children of the task manager. After some time, suppose that the policy is once again modified and the employee Sreevarshini is withdrawn from the same task of a manager.

The employee's Top most tasks should then be reset to sales, E-sales, Finance, E-Fin. Fig 5 shows an example for the Assign, Withdraw and Delete tasks, and their effect on the security tables.

Eventually, we note that the Tasks Tree itself may also be affected by the Delete operation. For instance, when a task is deleted, the tree is re-structured by moving all children of the deleted task, resulting they become siblings of that task. For example, suppose the Finance task is deleted then the E-Fin, M-Fin tasks should be connected directly to the task of a manager. For this purpose, we also provide a simple algorithm to rebuild the tree. It starts from the parent of the deleted node, extracts the sub-trees of its child nodes, and then attaches each one to the parent of the deleted task. The full tree can be re-structured if necessary. Fig 6 shows the task tree after deleting the Finance task.

Employees Tasks Table

Emp id	Emp Name	Task id	Task Name
1	Srivarshini	1	Production
1	Srivarshini	3	Finance
1	Srivarshini	5	sales
1	Srivarshini	6	e-sales
1	Srivarshini	8	e-Fin
1	Srivarshini	2	Manager
-----	----------	----	----------

Highest Tasks Table

Emp id	Task id	Task Name
1	1	Production
1	2	Manager

Fig 5 (a): After Executing: Assign sreevarshini to Manager

Employees Tasks Table

Emp id	Emp Name	Task id	Task Name
1	Srivarshini	1	Production
1	Srivarshini	3	Finance
1	Srivarshini	5	sales
1	Srivarshini	6	e-sales
1	Srivarshini	8	e-Fin
-----	----------	-----	----------

Highest Tasks Table

Emp id	Task id	Task Name
1	1	Production
1	3	Finance
1	5	sales
----	-----	------

Fig 5 (b): After Executing: Withdraw sreevarshini from Manager

Employees Tasks Table

Emp id	Emp Name	Task id	Task Name
1	Srivarshini	1	Production
1	Srivarshini	5	sales
1	Srivarshini	6	e-sales
1	Srivarshini	8	e-Fin
-----	----------	-----	----------

Highest Tasks Table

Emp id	Task id	Task Name
1	1	Production
1	5	sales
1	8	e-Fin
----	-----	--------

Fig 5 (c): After Executing: Drop Task Finance

Algorithm 1: For Deleting a Task:

1) Start
2) At first, locate the Task to be deleted
3) If (Deleted node = Head Task) i.e., at level 0 then
   3.1 Make the one of the child node as Head Task, and make the Siblings as Childs to the newly Head Task
   3.2 Automatically assign the values correspondingly after the deleted Task.
4) If (Deleted Task != Head Task) then
   4.1 If the Task is a leaf node, free the space of the deleted Task node and link the Tasks correspondingly
      4.1.1 Automatically assign the values correspondingly after the deleted Task
   4.2 If the deleted Task has one or more children, then make the children/children's of the deleted Task, as the Childs to the parent of the deleted Task and free the space of the deleted Task.
      4.2.1 Automatically assign the values correspondingly after the deleted Task
5) Stop

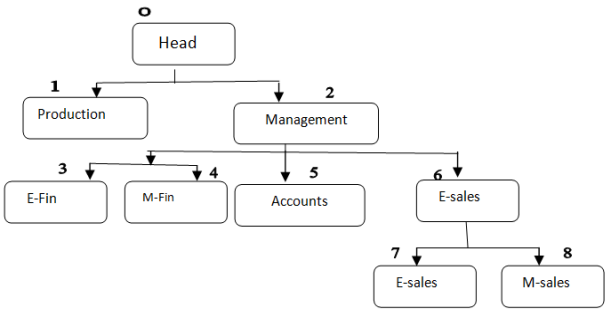

Fig 6: The Updated Task Tree

In addition to deleting a Task, in this approach it is possible to add a new node to the existing environment which is essential for dynamical environments. We have given an algorithm for adding a new node to the environment in Algorithm 2. In our example, we added a new node Marketing to the existing tree shaped structure then it is shown in the below Fig 7 & Fig 8.

Algorithm 2: For Adding a Task:

1) Start
2) At first, locate the Task to be added
3) If the newly added Task to be at level 0 i.e., as Head Task occurs in two cases:
   3.1 Replace the Existing Head Task with the Newly added Task
   3.2 Make the New Task as Head Task and make the Existing Head Task as child to the newly inserted Task
4) If the newly added Task to be at level 1:
   4.1 Add the new Task as child to the Head Task and make the Head Task as parent to the new Task and make the new Task as child to the head Task.
   4.2 Increment by 1, the values assigned to the Tasks after the newly inserted Task
5) If the newly added Task to be at level 2 or below
   5.1 Add the new Task as Child to the existing Tasks and make the newly added Task as children to the existing Task and make the existing Task as parent to the new Task.
   5.2 Increment by 1, the values assigned to the Tasks after the newly inserted Task
6) Now, get it the updated Tree
7) Stop

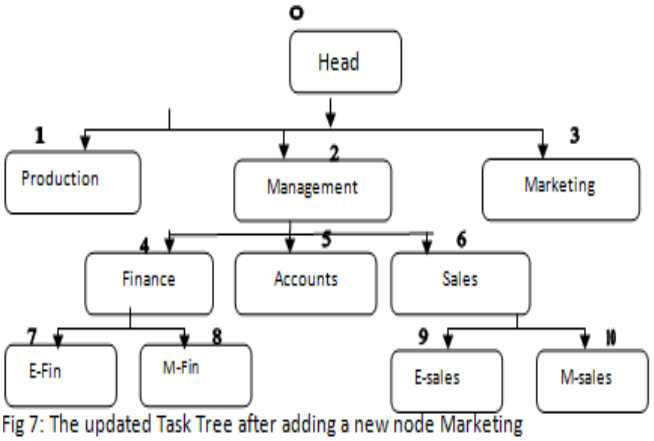

Fig 7: The updated Task Tree after adding a new node Marketing

Employees Tasks Table			
Emp id	Emp Name	Task id	Task Name
1	Srivarshini	1	Production
1	Srivarshini	3	Marketing
1	Srivarshini	6	sales
1	Srivarshini	7	E-Fin
1	Srivarshini	9	E-sales
-----	----------	-----	----------

Highest Tasks Table		
Emp id	Task id	Task Name
1	1	Production
1	3	Marketing
1	6	sales
1	7	E-Fin
----	-----	------

Fig 8: After Executing: Add the Task Marketing

VI. CONCLUSION

In this paper, we introduced a novel approach that is designed specifically to organize conflicting security tasks. In most complex distributed systems environment, an employee would likely have more than one task, and some of these tasks may have conflicted constraints. Our Novel approach for organizing the conflicting security Tasks (OCST) is constructed based on the well-known tree shaped structure. As an outcome of organizing the tasks in a way, constraints of multi-level hierarchies would be inherited by the successor tasks. Security administrators not only work in a familiar setting, but also verify security policies. We have also discussed the implementation of OCST using the class library (tree.hh). Finally, we shown how the tasks tree is re-shaped after applying the different operations such as assign, withdraw and delete tasks. The algorithms for adding and deleting a task in the dynamic environment are also provided. In conclusion, we deemed that the model presented in this work gives a good solution to the problem of overlapping/conflicting roles.

REFERENCES

1. Nicodemos Damianou, "A Policy Framework for Management of Distributed Systems", PhD thesis, Imperial College London, March, 2002.
2. Houbing Song, Glenn A. Fink and Sabina Jeschke, "Security and Privacy in Cyber-Physical Systems: Foundations, Principles, and Applications", Wiley-IEEE Press, Sept, 2017
3. Altamimi, "MORoles: An Abstract Hierarchy model for Managing Overlapping Security Roles", IJCSNS International Journal of Computer Science and Network Security, Vol.17 no.7, July 2017.
4. Altamimi, Ahmad and Eavis, Todd: "Securing Access to Data in Business Intelligence Domains". The International Journal on Advances in Security, vol 5, no 3 & 4. pp. 94 – 111. 2012.
5. Mariemma Yag¨ue. "Survey on xml-based policy languages for open environments",. Journal of Information Assurance and Security, 1(1):11–20, 2006.
6. Matteo Dell'Amico et. al., "Hipolds: a security policy language for distributed systems",. In Proceedings of the 6th IFIP WG 11.2 international conference on Information Security Theory and Practice: security, privacy and trust in computing systems and ambient intelligent ecosystems, WISTP'12, pp. 97–112, Berlin, Heidelberg, 2012. Springer-Verlag.
7. Sonia Jahid, Carl A. Gunter, Imranul Hoque, and Hamed Okhravi. "Myabdac: compiling XACML policies for attribute-based database access control", In Proceedings of the first ACM conference on Data and application security and privacy, CODASPY '11, pp. 97–108, New York, NY, USA, 2011. ACM.
8. Masoom Alam, Ruth Breu, and Michael Hafner. "Model-driven security engineering for trust management in sectet", Journal of Software, 2(1):4759, 2007
9. Skogsrud Halvard, R. Motahari-Nezhad Hamid, Benatallah Boualem, and Casati Fabio. "Modeling trust negotiation for web services", Computer, 42(2), pp. 54–61, 2009
10. PieroAndrea Bonatti et al., "Rule based policy representations and reasoning",. In Franois Bry and Jan Mauszyski, editors, Semantic .al for the Web, volume 5500 of Lecture Notes in Computer Science, pp.201–232. Springer Berlin Heidelberg, 2009.
11. Adrienne Porter Felt, Matthew Finifter, Joel Weinberger, and David Wagner. Diesel: applying privilege separation to database access. In Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security, ASIACCS '11, pp. 416–422, New York, USA, 2011. ACM.
12. Morris Sloman, "policy driven management for distributed systems", Journal of Network and Systems Management, Plenum Press, Vol. 2, No. 4, pp. 333-360, 1994.
13. Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein and Charles E. Youman, " Role-Based Access Control Models", IEEE Computer, Vol. 29, No. 2, pp. 38-47 Feb, 1996,.
14. Emil C. Lupu and Morris Sloman, "Towards A Role-Based Framework for Distributed Systems Management", *Journal of Network and Systems Management, Vol. 5, No. 1, 1997*
15. Ahn, G.-J and R. Sandhu "The RSL99 Language for Role-Based Separation of Duty Constraints. In Proceedings of the Fourth ACM Workshop on Role-Based Access Control, Fairfax, Virginia, USA, ACM Press, pp. 43-54, 28-29 Oct, 1999.

AUTHORS PROFILE



Dr. G S N Murthy is working as a Professor of CSE in Aditya College of Engineering, Surampalem. He completed his Ph.D (CSE) in Rayalaseema University, Kurnool, India. He has 23+ Years of teaching Experience and 5+ years of Research experience. He published various research articles in reputed International Journals and Conferences. He is reviewer for Various Scopus indexed journals and Editorial Board Member for Research India Group of Journals. He is the member for various professional bodies like IEEE, CSI and IAENG. His research work focuses on Data Mining, Image processing and Cyber security.



T. Veerraju is presently working as an Associate Professor in the Department of CSE, Aditya College of Engineering, Surampalem, India. He is having more than 22+ years of teaching experience and 3+ years of Research experience. He is a member of CSI and IAENG. His areas of interest include Cloud computing, Network security and Image Processing.



T. Satya Kumari is presently working as an Asst. Professor in the Department of CSE, Aditya College of Engineering, Surampalem, India. She is having more than 10 years of teaching experience. She is a member of CSI and IAENG. Her areas of interest include Data Mining and Network security.



V. Anantha Lakshmi is working as an Associate Professor in the Department of CSE, Aditya College of Engineering, Surampalem, India. She is having more than 10 years of teaching experience. She is a member of CSI and IAENG. Her areas of interest include Big Data and Network security.