Samyukta: A Unified Access Control Model using Roles, Labels, and Attributes

B. S. Radhika[1(✉)], N. V. Narendra Kumar[2], and R. K. Shyamasundar[3]

[1] Indian Institute of Information Technology Dharwad, Dharwad, India
radhikabs184@gmail.com
[2] Institute for Development and Research in Banking Technology, Hyderabad, India
nvnarendra@idrbt.ac.in
[3] Indian Institute of Technology Bombay, Mumbai, India

Abstract. Access control is one of the key mechanisms used for protecting system resources. While each of the existing access control models has its own benefits, it is difficult to satisfy all the requirements of a contemporary system with a single model. In this paper, we propose a unified model by combining three existing well-known models – Role-based Access Control (RBAC), Mandatory Access Control (MAC), and Attribute-based Access Control (ABAC) – in a novel way. The proposed model, named Samyukta, combines these three models in a modular way and uses them in a specific order so that the modules complement each other and we gain benefits of all three models. The widely used RBAC, with its roles, provides scalability, auditability, and easy management. With its labels, MAC provides Information Flow Control (IFC), and ABAC, with its attributes, provides flexible, context-aware, fine-grained access control. Along with these benefits, Samyukta also has advantages with respect to expressiveness, performance, and verifiability. We provide a relative comparison of our model with an existing model and also present a prototype implementation of Samyukta and demonstrate its efficiency.

Keywords: Access control · Role-based access control · Mandatory-based access control · Attribute-based access control · RBAC · MAC · ABAC

1 Introduction

Over the years, several access control models have been proposed, each with its own benefits and shortcomings. While adopting an access control model, features like usability, flexibility, auditability, ease of administration, scalability, etc., are considered.

While the existing models satisfy some of the requirements, no individual model can provide a comprehensive solution.

V. R. Badarla et al. (Eds.): ICISS 2022, LNCS 13784, pp. 84–102, 2022.
https://doi.org/10.1007/978-3-031-23690-7_5

RBAC is one of the most widely used models where access permissions are assigned to users through roles. It provides several benefits such as scalability, ease of administration, and auditability [23,32]. However, RBAC also suffers from shortcomings such as complex role-engineering, coarse-grained access control, lack of flexibility, and lack of support for dynamic-parameters such as time, location, etc. [15].

MAC is another well-known traditional model where access policy is specified in terms of subject and object labels. MAC supports information flow control which is essential in realizing properties like confidentiality, integrity, and privacy. However, MAC models are generally less flexible, coarse-grained, and lack support for dynamic attributes.

ABAC is an emerging model that specifies access rules in terms of attributes. This makes it a flexible and fine-grained model [15]. More importantly, its support for dynamic parameters based authorization is very useful. However, as the number of attributes and policy size increase, it becomes challenging to keep track of the permission assignments and ensure the enforcement of the intended access control.

From the above discussion, it is evident that each of the three popular models has its own advantages but fails to meet all the security requirements of contemporary systems. With the growing demand for more effective access control, numerous attempts have been made to enhance the existing models in various ways to achieve better authorization.

Several works have focused on integrating RBAC and ABAC. There are three broad ways of combining these two models [23]: (1) Dynamic roles: using attributes to dynamically map users to roles [2,17], (2) Attribute-centric: treating role as just another attribute, and (3) Role-centric: using attributes to constrain the permissions given by roles [20,30]. Among these classes, attribute-centric, loses the benefits of RBAC since it treats roles as attributes (and hence removing the abstraction provided by roles). The dynamic-roles approach simplifies the RBAC configuration and automates user-role mapping. However, by using attributes only during configuration (and not at run-time), it fails to capture the full benefits of the dynamic nature of attributes. The role-centric approach manages to benefit from both RBAC and ABAC by constraining permission of roles using dynamic attribute values. However, the existing role-centric works [20,30] require significant modifications to the RBAC model and have high performance overhead.

Attempts have also been made to combine RBAC and MAC [22,26,28,31]. Among them, some [22,26,31] emulate MAC using RBAC, some emulate RBAC using MAC [22]. The resulting configurations in these emulations are often too complex and unintuitive and therefore not practically viable for large systems. Works on unifying frameworks [3,19] generally focus on common specification of access control requirements rather than gaining benefits during deployment.

Thus, the current efforts to enhance the existing models are not adequate to provide effective access control. Most of them provide only ad-hoc solutions and therefore have limited applications. Moreover, many of these hybrid solutions fail

to capture their component models' advantages to the full extent. This is mainly because, most of these methods try to capture one model using the other and in the process lose the benefits of the original model. While combining multiple models, it is essential to retain the characteristic features of the component models in order to retain their inherent benefits. This paper proposes a unified model called `Samyukta` to address these problems. It uses a modular approach and provides an elegant way of combining the three models – RBAC, MAC, and ABAC – that is useful both at a specification and at implementation level. Our contributions in the paper include:

- Presenting a unified model referred to as, `Samyukta`,(which means *unified* in Sanskrit) that combines RBAC, MAC, and ABAC in such a way that we can essentially unify the benefits of each of the models.
- Developing a prototype implementation of `Samyukta` and demonstrated its efficiency.

The remainder of the paper is organized as follows: In Sect. 2, we present a brief description of the MAC model used in `Samyukta`. Section 3 reviews the related works. In Sect. 4, we present the motivation, followed by details of `Samyukta` in Sect. 5. In Sect. 6, we discuss the effectiveness of `Samyukta` and Sect. 7 discusses the experimental analysis. Finally, Sect. 8 concludes the paper.

2 Related Work

In this section, we highlight some of the important works that use RBAC, MAC, and ABAC in various combinations.

Kuhn et al. [23] and Cyne and Weil [9] have discussed different ways of combining RBAC and ABAC. As discussed in Sect. 1, the role-centric approach is more effective in integrating the benefits of these two models. RABAC [20] and AERBAC [30] are two such role-centric solutions. RABAC uses Permission Filtering Policy (PFP) to constrain the permissions of the roles. AERBAC adds attribute-based conditions to the RBAC authorization. Both the methods require modification to the standard RBAC. Several RBAC extensions such as temporal [25], spatial [10], and spatio-temporal [1] have been proposed to include dynamic parameters like time and location in RBAC authorization. Researchers have also worked on converting RBAC policies to ABAC [8] and ABAC policies to RBAC [4] in order to gain benefits of the target model. C-ABAC [12] proposes a formal model of ABAC using category- based meta-model of access control (CBAC) [3]. Here permissions are assigned to categories of users rather than to individual users. For this, along with the principal, resource, and environment attributes, it also uses category attributes and access decision is taken based on these attribute values. Although, grouping users into categories (which is similar to RBAC) is an improvement over regular ABAC, it doesn't provide all the benefits of RBAC and also doesn't provide information flow control. The latter is crucial for privacy.

Works have also been proposed to combine RBAC and MAC. Sandhu [31] proposed a method to emulate lattice-based access control using role hierarchies and constraints. Osborn [27], Gofman et al. [14], and Stambuli et al. [33] have worked on information flow control in RBAC to achieve the benefits of MAC using RBAC configuration. Tuval and Gudes [34] attempt to analyze the information flows in a given RBAC configuration and resolve flow conflicts. These solutions work for only smaller systems with few roles and objects. As the number of roles and objects increase, the analysis becomes complex and cumbersome.

Jin et al. [19] propose a unified attribute-based model that can help specify DAC, MAC, and RBAC policies using ABAC. However, once MAC labels and RBAC roles are represented as attributes, their inherent benefits such as auditability and easy management will be lost. Barker [3] has proposed a unified meta-model that aims to provide a basis for a common specification of access control requirements. Based on this model, Kafura and Gracanin [21] have proposed a meta model for information flow control. Jajodia et al. [18] proposed a unified framework that can enforce multiple access control policies. These models and frameworks are flexible and can model various existing systems. However, they focus only on the specification of requirements rather than ways to achieve effective authorization.

Thus, we can see that several attempts have been made to integrate the existing models for better access control. However, the current solutions are not adequate to support an effective and practical implementation. In the next section, we illustrate the need for an unified model with an example.

3 Need for a Unified Model

In this section, we motivate the need for a unified model through a simple example.

Example 1. Consider an organization with two types of users: *clerk* and *manager*. There are two users: *cl*, a *clerk* and *mg*, a manager. A clerk is responsible for bookkeeping of financial transactions that are stored in a file named $txnFile$ and therefore has read and write permissions for the file. A manager is responsible for handling business management and has read and write permissions for a management file named $mgmtFile$. The $mgmtFile$ contains sensitive business information that should not be revealed to a *clerk*. When needed, a *manager* can also perform the $clerk's$ tasks and therefore has all the associated permissions. Furthermore, the organization has two constraints on the permissions (i) a transaction file can be written only during the working hours of working days, (ii) a management file can be written only from within the office.

The above example includes various security requirements that are commonly seen in organizations. It has access permissions based on the user's job or position in the organization. It needs to protect sensitive information against leaks. It also needs to control certain permissions based on various characteristics of the entities involved in the access and access environment.

The requirements based on the users job description can be well captured by RBAC. A simple RBAC configuration for the above example includes two users $\{cl, mg\}$, two objects $\{txnFile, mgmtFile\}$, two roles $\{clerk, manager\}$. The role *manager* is higher up in the role hierarchy and therefore inherits all the permissions assigned to *clerk*. The complete configuration with user-role assignment (UA), permission-role assignment (PA), and role hierarchy (RH) is shown in Fig. 1. This configuration is easy to understand as it clearly represents the organizations structure. The organization can easily add any new employees by simply assigning them to the suitable role. Also, the organization can easily find the set of permissions available to a user simply by first getting the active roles of the user and then by finding the set of permissions mapped to those roles. This makes auditing easy.

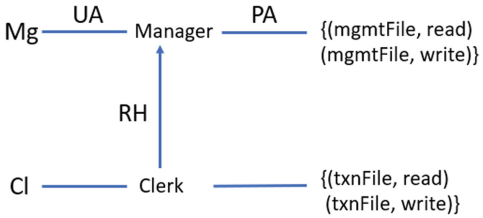

Fig. 1. RBAC configuration

However, this configuration doesn't prevent leak of information from $mgmtFile$ to cl. The leak is possible through indirect information flow caused when mg (as *manager*) reads $mgmtFile$ and writes the content to $txnFile$ which can then be read by cl (with role *clerk*). This indirect information flow is shown in Fig. 2.

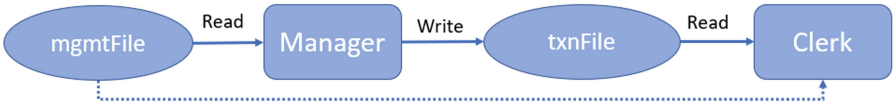

Fig. 2. Indirect information flow

Also, the above configuration has not covered the constraints specified in terms of time, day, and location. RBAC fails to support authorizations based on such dynamic values easily. Since all the permissions are mainly decided statically during role creation and user-role mapping, one way to support such parameters is to create multiple roles for each of the values of these parameters. Thus, for considering all the three parameters (time, day, location), each with two possible values (working hour, non-working hour, working day, non working day, office, out-of-office), in the worst case, we may need to create 8 roles each

for *manager* and *clerk*, leading to role-explosion. Moreover, any slight change in these constraints may require changes in the role configuration making the authorization system cumbersome to manage.

The requirements related to confidentiality and integrity that involve information flow control are generally captured by MAC models. Let us consider a simple MAC configuration for Example 1 using a classic multi-level security (MLS) model, BLP. It would involve using two labels, l_1 and l_2 where $l_2 > l_1$. l_1 is assigned to the $txnFile$ and cl and l_2 is assigned to $mgmtFile$ and mg (since $mgmtFile$ is confidential and is at a higher security level). BLP allows only information flows from lower to higher levels through its *no-read-up* and *no-write-down* rules. As a result, the manager mg with level $l2$ will not be allowed to write to $txnFile$. Thus preventing flow from $mgmtFile$ to cl. However, notice that this configuration doesn't really capture the job descriptions and organization's structure. Also, such basic MAC models are not flexible enough for practical usage in mainstream systems. Furthermore, similar to RBAC, MAC models also don't support dynamic parameters.

ABAC is known for its fine-granularity and flexibility. An ABAC configuration for Example 1 can be created by using the attributes $role, object\_type, location, time, day$. With these attributes, a policy can be defined as shown in Fig. 3. Note that we can also use an attribute *level* to capture the simple MLS rules discussed above. However, to keep the policy simple and intuitive, we have not included it in the policy.

ABAC Policy 1

1. $(read, (object\_type(o) = transaction \wedge level(o) \leq level(s)))$
2. $(write, (object\_type(o) = transaction \wedge level(s) \leq level(o) \wedge day = working\_day \wedge time = working\_hour))$
3. $(read, (role = manager \wedge object\_type(o) = management \wedge level(o) \leq level(s)))$
4. $(write, (role = manager \wedge object\_type(o) = management \wedge level(s) \leq level(o) \wedge location = office))$

Fig. 3. ABAC policy for Example 1

With such attribute-based policy, we can specify precise access rules. These rules are easy to change as per the changing requirements. However, notice that it is not easy to understand the organization's structure and retrieve permissions assigned to a user (auditing) easily from the policy, especially when the policy size increases.

Thus, as the policies get larger, their management and auditing (involves enumerating over all the rules) becomes very difficult.

Table 1. Advantages and shortcomings of RBAC, MAC, and ABAC

	Advantages	Shortcomings
RBAC [9,23,32]	– Captures organization's structure – Scalable – Easy to manage – Auditable	– Complex role engineering – Role explosion – Coarse-grained – Lack of flexibility – Lack of support for dynamic parameters
MAC [16,24]	– Provides IFC – Scalable	– Lack of flexibility – Lack of support for dynamic parameters – Coarse-grained
ABAC [9,15]	– Fine-grained – Flexible – Supports dynamic parameters	– Difficult to audit – Difficult to scale

Note that the above configurations are the straightforward, intuitive usage of each of the three models.

The RBAC, MAC, and ABAC models are expressive enough to emulate the other models and capture the aspects that are not attributed to them in the above description. For example, RBAC can be configured to support IFC [31,33] and finer-grained access [2,20,30], ABAC can be configured to support roles and IFC [19], MAC can emulate RBAC [22]. However, such configurations are unintuitive and cumbersome to manage. Using three models instead of a single model naturally incurs performance overhead. On of our major goals is to minimize this overhead caused while combining these models.

Table 1 summarizes the benefits and shortcomings of each of the three models. From the table, it is evident that while each of the three models has its own strengths, it is difficult to cover all the requirements using any one of them effectively. Trying to capture all the requirements with any one of these models will result in unintuitive and hard-to-manage policies. Using three different policies help in separating the requirements based on their nature and create modular policies. So it would be more effective to use these models without changing their inherent characteristics and combine them in such a way as to we gain best of each of the component models without significant performance overhead. In this paper, we combine these three models – RBAC, MAC, and ABAC– using a modular approach to get the best of each of the models and provide comprehensive access control.

4 Preliminaries

Basic authorization concepts of RBAC [32] and ABAC [15] are well-known. Hence, for the purpose of brevity, we refer the reader to standard expositions. In this section, we briefly describe a relatively new lattice-based MAC model called Readers-Writers Flow Model (RWFM) that is used in Samyukta.

4.1 Readers-Writers Flow Model

Readers-Writers Flow Model (RWFM) [24] is a decentralized information flow model. It is complete with respect to the Denning's model [11,24] and can be used to provide both confidentiality and integrity. It supports dynamic labeling and downgrading. Also, it can capture several well-known models such as Bell-LaPadula (BLP) [5], Biba [6], Chinese Wall [7]. These features make RWFM better suited for mainstream application. So, in Samyukta, we use RWFM as the MAC module. Its labeling and access rules are provided below:

Labeling: Let S and O be the set of subjects and objects in the system respectively. An RWFM label (*RW Class*) is defined as a triplet (s, R, W). Where $s \in S$ denotes the owner of the information in the class. $R \in 2^S$ denotes the set of subjects which can read the objects of the class. $W \in 2^S$ denotes the set of subjects that can write or have influenced the class.

Since objects are passive entities, in the RWFM implementation their labels are static, i.e., once a label is assigned to an object, it remains unchanged. Whereas in the case of subjects, which are dynamic in nature, the labels can change. When a subject is created, it has gained no information. Such a subject is positioned at the bottom of the lattice. Which means, the readers of the newly created subject includes all the subjects in the system and the only writer is just the subject itself (as no other subject has yet influenced it). Thus, on creation, a subject's *readers* (R) is set to S and *writers* (W) is set to the subject itself. After this initialization, the label can change according to the information gain as the subject performs various operations.

Access Rules: With the above labeling model in place, access rules of RWFM are specified as follows:

- *Read(slabel, olabel)*- RWFM provides flexibility by supporting reading-up i.e a subject s can read an object o that is higher up in the lattice provided $owner(s) \in R(o)$. Before allowing the access, the label of the subject is modified and moved up the lattice to reflect the information gain. So, when a subject s_1 with label *slabel* of the form (s_1, R_1, W_1) reads an object o_1 with label *olabel* of the form (s_2, R_2, W_2), its label is changed as follows: $(s_1, R_1, W_1) \bigoplus (s_2, R_2, W_2) = (s_1, R_1 \cap R_2, W_1 \cup W_2)$.
- *Write(slabel, olabel)*- A subject s_1 with label *slabel*:(s_1, R_1, W_1) is allowed to write an object o_1 with label *olabel*:(s_2, R_2, W_2) if $s_1 \in W_2$ and $R_1 \supseteq R_2$ and $W_1 \subseteq W_2$.

5 Samyukta: A Unified Access Control Model

In this section, we present our unified model Samyukta. We first discuss the model's formal specification. We then describe the request flow in Samyukta and its authorization procedure.

5.1 Formal Specification

The specification of Samyukta consists of basic access control components and components relevant to RBAC, MAC, and ABAC models. The basic components include:

- $U-$ a set of authorized users in the system.
- $S-$ a set of subjects/sessions in the system.
- $O-$ a set of objects in the system.
- $Op-$ a set of operations (actions) that can be performed on the objects.
- $owner(e : S \cup O) \rightarrow U$ is a function that maps an entity e (subject or object) to its corresponding owner.

Components relevant to RBAC are:

- $R-$ denotes a set of roles where a role represents a job function in the organization.
- $RH \subseteq R \times R$ is a partial order on R that denotes the role hierarchy where $(r_1, r_2) \in RH$ implies that r_1 is higher up in the hierarchy and therefore inherits all the permissions of r_2. It can also be denoted as $r1 > r2$.
- $PERMS \subseteq O \times Op$ is the set of permissions that are allowed in the system.
- $UA \subseteq U \times R$ is a many to many mapping from users to roles.
- $PA \subseteq PERMS \times R$ is a many to many mapping from permissions to roles.
- $active\_roles(s : S) \rightarrow 2^R$ is a function that maps session s to a set of roles that are active in the session.
- $avail\_session\_perm(s : S) \rightarrow 2^{PERMS}$ is function that maps a session s to the set of permissions available in that session.
- C represents the constraints used with respect various RBAC configuration components such as UA, PA, RH, etc.

Components relevant to MAC are:

- L represents the set of security labels.
- $\leq \subseteq L \times L$ represents permissible information flows. $l1 \leq l2$ where $l1, l1 \in L$ indicates that information in $l1$ can flow to $l2$.
- $\bigoplus : L \times L \rightarrow L$ represents the label combination (Least Upper Bound) operation.
- $\lambda : S \cup O \rightarrow L$ is a function that assigns labels to subjects and objects.
- $FlowDirection : Op \rightarrow \{in, out, both, none\}$ is a function that denotes the type of information flow caused by an operation $op \in Op$. Here *in* denotes information flow from object to subject (as in case of *read*), *out* denotes flow from subject to object (as in case of *write*), *both* denotes flow in both the directions, and *none* denotes no information flow.

Finally, components of the ABAC module are as follows:

- $UAtt$ represents the set of user attribute names.
- $SAtt$ represents the set subject attribute names.
- $OAtt$ represents the set of object attribute names.
- $EAtt$ represents the set of environment attribute names.
- $Range(a)$ denotes a function that maps an attribute $a \in UAtt \cup SAtt \cup OAtt \cup EAtt$ to the set of all possible values for a.
- $\forall ua \in UAtt,\ \exists f_{ua} : U \rightarrow Range(ua)$ represents a function that maps a user to its value for attribute ua.
- $\forall sa \in SAtt,\ \exists f_{sa} : S \rightarrow Range(sa)$ represents a function that maps a subject to its values for attribute sa.
- $\forall oa \in OAtt,\ \exists f_{oa} : O \rightarrow Range(oa)$ represents a function that maps an object to its value for attribute oa.
- $\forall ea \in EAtt,\ \exists ea \mapsto Range(ea)$ represents a mapping from environment attribute ea to its current value.
- $Access\_Rules$ denotes a set of rules of the form $(op, Cond)$ where $op \in Op$ and $Cond$ corresponds to conditions specified in terms of attributes of users, subjects, objects, and environment.

This specification encompasses all the components necessary for RBAC, MAC, and ABAC models. We can divide the security requirements among the three models and specify three policies. Though we have the overhead of specifying three different policies, we get the benefits offered by each of these policies and having a modular policy gives the advantage of modifying one policy without affecting the other. Note that as per security requirements, we can also omit the components that are not needed in a specific system. For example, if a system does not require IFC, we can omit the MAC components, if a system requires separate roles for *working hours* and *non working hours*, then we can omit the *time* attribute. Thus with this generic and expressive specification, we can encode several of the existing access control models.

5.2 Request Flow in Samyukta

As mentioned, we use a modular approach where the modules are independent of each other so that they can be configured and managed separately. The three modules are ordered i.e., an access request is sent to the component modules in the specific order of RBAC$\longrightarrow$ MAC $\longrightarrow$ ABAC as shown in Fig. 4. The RBAC module specifies the maximum set of permissions available to a subject. The MAC module, which follows the RBAC module, allows the subset of accesses that satisfy the MAC rules and therefore are flow-secure. Here we use RWFM for MAC and in the rest of the paper we use RWFM and MAC interchangeably. The accesses that are allowed both by RBAC and MAC are forwarded to ABAC module which further applies constraints based on the attribute values.

The order in which the modules are used plays a vital role in harnessing the benefits of the individual modules. We use RBAC as the first modules as it is

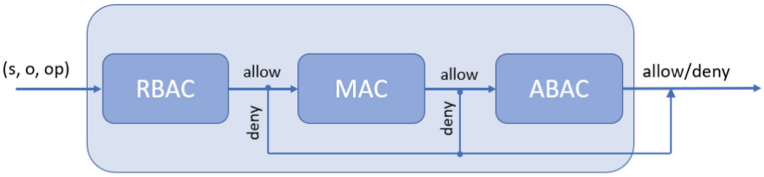

Fig. 4. Request flow through the modules

easy and straightforward to capture the organization's structure with it. It also provides easy management, scalability, and auditability. Hence we use RBAC to assign maximum set of permissions to the users and once RBAC captures the organization's structure through its roles and role hierarchies, MAC and ABAC modules are used to fine-tune these authorizations based on information flow and attribute based requirements. To reap the full benefits of fine-granularity, and dynamic nature of ABAC, it is used as the last module. Thus, MAC serves as the second module. Using MAC after RBAC enables use of the RBAC policy as the basis for labelling of subjects/objects for RWFM. This helps in keeping track of information flows and prevent information leaks caused by RBAC permissions.

The RBAC configuration in Fig. 1 clearly captures the job-based requirements of Example 1. However, as discussed, it fails to protect the sensitive information in *mgmtFile*. Using RWFM after RBAC helps in fulfilling such requirements by identifying and preventing indirect flows while providing adequate flexibility. The RWFM labels for objects of Example 1 would be (note that here we use the labels at the granularity of roles instead of subjects):

- $mgmtFile$: $\{manager, \{manager\}\{manager\}\}$ The $mgmtFile$ can be read and written by only $manager$. Therefore, the $readers$ and $writers$ components of the label have single element.
- $txnFile$: $\{clerk, \{manager, clerk\}\{manager, clerk\}\}$ Since the $txnFile$ can be read and written by both $manager$ and $clerk$, both are part of the $readers$ and $writers$ components of the label.

When a subject s corresponding to Mg is created, its label will be initialized to $\{manager, \{manager, clerk\}\{manager\}\}$. With these labels set, when s reads $mgmtFile$, as per the RWFM read rule, its label changes to $\{manager, \{manager\}, \{manager\}\}$. Now when s tries to write to $txnFile$, RWFM write rule fails ($R(s) \not\supseteq R(txnFile)$) preventing the transitive flow, and thereby fulfilling the requirement. Thus, with dynamic labeling, RWFM effectively prevents indirect information flows and also proves to be more flexible than the traditional models like BLP [24].

Finally, the ABAC module deployed at the end provides authorization at a finer granularity. For Example 1, with most of the requirements covered by RBAC and MAC, the ABAC has to cover only the two constraints on write operation. This can be easily specified by using the attributes $object\_type, location$, $time$, and day. With these attributes, a policy can be defined as shown Fig. 5. ABAC

module is useful especially in systems where access decisions need to be taken based on continuous-valued attributes such as time, temperature, etc., mainly because it is difficult capture such authorization with other models.

ABAC Policy

1. $(write, (object\_type(o) = transaction \wedge day = working\_day \wedge time = working\_hour))$
2. $(write, (object\_type(o) = management \wedge location = office)$

Fig. 5. ABAC policy in Samyukta

Using the ABAC model as the last model is also useful with respect to performance. Compared to RBAC and MAC, ABAC is performance intensive. As ABAC is used mainly to constrain the earlier models' permissions, its policy size would be significantly smaller (compared to a standalone ABAC policy such as the policy in Fig. 3) and therefore requires less computation. Also, as ABAC processes only those requests allowed by the previous two models, it avoids unnecessary evaluations of ABAC rules for all the requests and helps achieve good performance.

Thus, whether an access is permitted by Samyukta, depends on whether the subject's role is authorized to perform the access, whether the information flow caused by the action is permitted, and finally, whether all the dynamic and fine-grained constraints are satisfied.

Samyukta also aids the models to complement one another effectively. The fine-grained access control of ABAC overcomes the coarse-granularity in RBAC and RWFM. RBAC's drawbacks of role explosion, lack of flexibility, and dynamic parameters are overcome with the flexible and dynamic ABAC. Lack of information flow control in RBAC is overcome with MAC. ABAC's drawbacks in terms of scalability, policy management, and performance are overcome with RBAC and MAC.

As discussed above, the order of models used in Samyukta has several benefits that are useful in most of the cases. However, in certain cases, depending on the granularity of the attribute, and the access requirements, we can also use ABAC before RBAC. For example, in the scenario discussed in Example 1, if all the accesses are allowed only within the office, then we can check the *location* attribute first. We can check other modules only if value of the attribute *location* is *office*. This way, we can avoid evaluation of other modules and speed of the authorization.

5.3 Authorization Procedure

Now, we describe how an access request is processed by `Samyukta`. Given a `Samyukta` configuration and an access request of the form (s, o, op) where s is a subject id, o is an object id, and op is an operation, `Samyukta` first sends the request to the RBAC module followed by the MAC and ABAC modules. If any of these modules denies the access, the request is not forwarded to the next modules, and it is denied by `Samyukta`. A request is authorized only if it is authorized by all three modules. This procedure is described in Algorithm 1. The authorization in the individual modules are described below:

Algorithm 1: SamyuktaAuthorization

```
Input  : A Samyukta configuration and an access request Req of the form
         (s, o, op)
Output: Authorization decision of Samyukta: ALLOW or DENY
if RBAC_Authorization(Req) then
    if MAC_Authorization(Req) then
        if ABAC_Authorization(Req) then
            return ALLOW
        end
    end
end
return DENY
```

RBAC Authorization: The authorization of RBAC is simple. When an access request is received, the module extracts the subject's active roles and checks if any of these roles have the requested permission. If so, the access is allowed else denied. As the procedure is fairly straightforward, we omit the algorithm for brevity.

MAC Authorization: Here we first extract the RWFM labels of subject and object using the corresponding IDs and then depending on the direction of the information flow, we apply the RWFM access rules discussed in Sect. 4. The authorization procedure used in this module is given in Algorithm 2. Here, *Read()* and *Write()* functions apply the RWFM read and write rules respectively and return *True* (success) or *False* (failure).

ABAC Authorization: When the ABAC module receives a request, it extracts the associated attributes and then evaluates the access rules applicable for the requested operation (this is done by *eval()* function that evaluates the conditions of the rule with respect the extracted attributes). If the conditions are satisfied, the access is allowed otherwise denied. This procedure is described in Algorithm 3. Note that since ABAC is mainly used to constrain the permissions given by the previous modules, if no rule applies to a request, the request will be permitted by ABAC. Therefore, if there are no constraints on an operation (as in case of the read operation in Example 1), then we do not have to specify any ABAC rules for it and such operations can be allowed without evaluating any ABAC rules.

Algorithm 2: MAC_Authorization

```
Input  : An access request Req of the form (s, o, op)
Output: Authorization decision: ALLOW or DENY
slabel = getSubjectLabel(s)
olabel = getObjectLabel(o)
flow_dir = getFlowDirection(op)
if flow_dir = none then
    return ALLOW
else if flow_dir = in then
    if Read(slabel, olabel) then
        return ALLOW
    end
else if flow_dir = out then
    if Write(slabel, olabel) then
        return ALLOW
    end
else if flow_dir = both then
    if Read(slabel, olabel) AND Write(slabel, olabel)) then
        return ALLOW
    end
return DENY
```

Algorithm 3: ABAC_Authorization

```
Input  : An access request Req of the form (s, o, op)
Output: Authorization decision of: ALLOW or DENY
u = owner(s)
uatt = getUAttributes(u)
satt = getSAttributes(s)
oatt = getOAttributes(o)
eatt = getEAttributes()
foreach (op, cond) in AccessRules do
    if not eval(cond) then
        return DENY
    end
end
return ALLOW
```

6 Effectiveness of Samyukta

Here, we demonstrate the effectiveness of Samyukta by comparing it with the existing RBAC-ABAC hybrid model called RABAC [20]. RABAC is a role-centric model that extends the NIST RBAC model [13,32] with attributes and permission filtering policies (PFP). PFPs constrain the available set of permissions based on user and object attributes. Conceptually, it is similar to the way

`Samyukta` combines RBAC and ABAC, albeit with MAC in between them and different enforcement techniques.

In [20], the authors use a hospital example with two main roles, Doctor, and VisitDoc (visiting Doctor). Doctors are allowed to read their patients' records at any time. VisitDoc is only allowed to read authorized documents which are revealed for collaboration purpose with other hospitals and the request will only be approved during working hours and if made from any certified devices. In addition, visiting doctors from other hospitals are only allowed to view authorized documents pertaining to the projects they participate in. For the illustration, we use three objects $\{o1, o2, o3\}$ where $\{o1, o2\}$ are patient records and $o3$ is an *AuthorizedDoc*. To configure this example, RABAC uses two filtering functions, one for patient records and one for authorized docs. These filters are applied to all the requests, irrespective of whether the basic RBAC rules allow the request or not.

In case of `Samyukta`, the RBAC module uses PA to assign the maximum set of permissions to roles. ABAC applies its *Access_Rules* on the requests authorized by RBAC to constrain the permissions of RBAC. Some of the important components of Samyutka for this scenario are given in Fig. 6. Here we use three user attributes, *doctorof* which represents the patients of a doctor, *uproj* which represents the projects that a user is associated with, and *device* which corresponds to the device used by the user. It also uses three object attributes, *object_type*, which specifies the type of the object, *recordof*, which specifies the patient to whom the record belongs, and *oproj*, which represents the project (where object is a project file). Finally, it uses an environmental attribute *time* that indicates the time at which an access request has been made.

From the above illustration, we can observe that RABAC requires modifications to the standard RBAC model and requires processing attribute-based rules for every access request incurring performance overhead. Whereas in case of `Samyukta`, we can use the standard RBAC without any modification and therefore get the benefits that are inherent to the model and use attribute-based rules for only those requests that satisfy the RBAC rules which leads to better performance.

Note that here we have not used components related to the MAC model as there is no explicit IFC requirement in the example.

Configuration 1: Configuration for Hospital Scenario using Samyukta

- $PA : \{(doctor, (o1, view)), (doctor, (o2, view)), (visitDoc, (o3, view))\}$
- $UAtt = \{doctorof, uproj, device\}$ $OAtt = \{object\_type, recordof, oproj\}$ $EAtt = \{time\}$
- $Range(object\_type) = \{PatientRecord, AuthorizedDoc\}$
- $Range(uproj) = Range(oproj) = Projects$
- $Range(recordof) = Range(doctorof) = Patients$
- $Access\_Rules = \{(view, (object\_type(o) = PatientRecord \wedge recordof(o) \in doctorof(s)),$ $(view, (object\_type = AuthorizedDoc \wedge device(s) = certified \wedge oproj(o) \in uproj(u) \wedge$ $time = working))\}$

Fig. 6. Configuration for the hospital scenario

6.1 Merits of Samyukta

Some of the major advantages of Samyukta are briefed below:

- It is modular and hence supports independent configuration and management of each of the models.
- Since we don't modify the component models, it is possible to effectively use the existing standards, best-practices, verification techniques/tools for the modules.
- It gains benefits of all the component models.
- It is generic and can capture a spectrum of existing models.

7 Experimental Analysis

In this section, we demonstrate the performance of the prototype implementation of Samyukta. We have implemented Samyukta using Python 3.8 on a system running Ubuntu 20.04 with 16 GB RAM. RBAC and MAC implementations follow from existing implementations using simple data structures and set operations. ABAC implementation follows from a XACML-based design [29] and uses JSON for request and access rules specification.

As discussed, using the modules in the specified order where each module filters out the requests that it authorizes has a performance advantage. This is especially useful with respect to the ABAC module as evaluating attribute-based rules incur relatively high performance overhead. Unlike in previous works such as [20] and [30], Samyukta evaluates the attribute-based rules only for the requests that are permitted by both RBAC and MAC module and therefore saves significant computation.

To demonstrate the performance benefit of filtering the requests, we perform the following two executions and compare their execution times: (1) Without filtering: here we pass a set of requests *Reqs* to the three modules separately so that each module processes all the requests in *Reqs* (2) With filtering: here we process the requests in *Reqs* using Samyukta which uses the modules in sequence and requests are filtered out at each step.

We perform the experiment on three synthetic datasets (DS) shown in Table 2. We run each experiment 10 times and record the average time taken. We have set $|Reqs| = 200$ in each run to get a good mix of requests that go through all the modules when we pass them through the modules in sequence. The experiments' results for the three datasets for with and without request filtering are given in Table 3. From the results, it is evident that Samyukta gains significant performance by using the filtering approach.

Table 2. Datasets

	DS1	DS2	DS3
Users	5	10	20
Subjects	50	100	200
Objects	50	100	200
Roles	5	10	20
UAtt	5	10	20
SAtt	5	10	20
OAtt	5	10	20
EAtt	5	10	20

Table 3. Execution times for with and without request filtering

Dataset	Execution time (in microseconds)	
	Without filtering	With filtering
DS1	2879.51	23.37
DS2	3727.87	77.85
DS3	3794.10	98.30

Table 4. Execution times for different module orderings

Dataset	Execution time (in microseconds)					
	Samyukta	MRA	RAM	MAR	AMR	ARM
DS1	23.37	23.52	34.24	1248.26	2821.56	2893.72
DS2	76.05	77.85	132.12	1888.55	3741.21	3721.30
DS3	95.17	98.30	136.89	179.19	3795.56	3304.41

Also, among the 6 possible orderings of the modules possible, the order used by Samyukta provides better performance. We demonstrate this by comparing their execution times for processing the three datasets in Table 2 by Samyukta, MRA (MAC-RBAC-ABAC), RAM (RBAC-ABAC-MAC), MAR (MAC-ABAC-RBAC), AMR (ABAC-MAC-RBAC), and ARM (ABAC-RBAC-MAC). The execution times for different module orderings are shown in Table 4 (the approach used is same as that of the previous experiment). From the table it is clear that the order used by Samyukta provides better performance than the other orderings.

8 Conclusions

Providing easy-to-use, effective access control is crucial in protecting system resources from intended or unintended misuse. It is difficult to achieve all the

desired authorization requirements in a contemporary system with a single access control model. In this paper, we have presented a unified model, Samyukta, that combines RBAC, MAC, and ABAC in such a way that the user can benefit from the best of all these models. The proposed solution is simple, elegant, and flexible so that the user can implement the solution as per their security requirements. Our experimental analysis has demonstrated that the filtering approach and the ordering of the modules used in the proposed model significantly improve its performance making it viable for practical usage.

Acknowledgement. The work presented in this paper was done at the Indian Institute of Technology Bombay and was supported by the Information Security Research and Development Centre, Ministry of Electronics and Information Technology, Government of India.

References

1. Abdunabi, R., Al-Lail, M., Ray, I., France, R.B.: Specification, validation, and enforcement of a generalized spatio-temporal role-based access control model. IEEE Syst. J. **7**(3), 501–515 (2013)
2. Al-Kahtani, M.A., Sandhu, R.S.: A model for attribute-based user-role assignment. In: CSAC, pp. 353–362 (2002)
3. Barker, S.: The next 700 access control models or a unifying meta-model? In: SACMAT Proceedings, pp. 187–196 (2009)
4. Batra, G., Atluri, V., Vaidya, J., Sural, S.: Deploying ABAC policies using RBAC systems. J. Comput. Secur. **27**(4), 483–506 (2019)
5. Bell, D.E., LaPadula, L.J.: Secure computer systems: mathematical foundations. Technical report MTR-2547-VOL-1, MITRE Corp., Bedford, MA (1973)
6. Biba, K.J.: Integrity considerations for secure computer systems. Technical report. MTR-3153-REV-1, MITRE Corp., Bedford, MA (1977)
7. Brewer, D.F.C., Nash, M.J.: The Chinese wall security policy. In: Proceedings of IEEE Symposium on Security and Privacy, pp. 206–214 (1989)
8. Chakraborty, S., Sandhu, R., Krishnan, R.: On the feasibility of RBAC to ABAC policy mining: a formal analysis. In: SKM, Proceedings, pp. 147–163 (2019)
9. Coyne, E., Weil, T.R.: ABAC and RBAC: scalable, flexible, and auditable access management. IT Prof. **15**(3), 14–16 (2013)
10. Damiani, M.L., Bertino, E., Catania, B., Perlasca, P.: GEO-RBAC: a spatially aware RBAC. ACM Trans. Inf. Syst. Secur. **10**(1), 2 (2007)
11. Denning, D.E.: A lattice model of secure information flow. Commun. ACM **19**(5), 236–243 (1976)
12. Fernández, M., Mackie, I., Thuraisingham, B.M.: Specification and analysis of ABAC policies via the category-based metamodel. In: Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy, CODASPY 2019, pp. 173–184. ACM (2019)
13. Ferraiolo, D.F., Sandhu, R.S., Gavrila, S.I., Kuhn, D.R., Chandramouli, R.: Proposed NIST standard for role-based access control. ACM Trans. Inf. Syst. Secur. **4**(3), 224–274 (2001)
14. Gofman, M.I., Luo, R., Solomon, A.C., Zhang, Y., Yang, P., Stoller, S.D.: RBAC-PAT: a policy analysis tool for role based access control. In: Proceedings TACAS, pp. 46–49 (2009)

15. Hu, V.C., Ferraiolo, D., Kuhn, R., Friedman, et al.: Guide to attribute based access control (ABAC) definition and considerations. NIST Spec. Pub. 800 (162) (2013)
16. Hu, V.C., Kuhn, D.R., Xie, T., Hwang, J.: Model checking for verification of mandatory access control models and properties. Int. J. Softw. Eng. Knowl. Eng. **21**(1), 103–127 (2011)
17. Huang, J., Nicol, D.M., Bobba, R., Huh, J.H.: A framework integrating attribute-based policies into role-based access control. In: SACMAT, pp. 187–196 (2012)
18. Jajodia, S., Samarati, P., Sapino, M.L., Subrahmanian, V.S.: Flexible support for multiple access control policies. ACM Trans. Database Syst. **26**(2), 214–260 (2001)
19. Jin, X., Krishnan, R., Sandhu, R.S.: A unified attribute-based access control model covering DAC, MAC and RBAC. In: DBSec 2012 Proceedings, pp. 41–55 (2012)
20. Jin, X., Sandhu, R.S., Krishnan, R.: RABAC: role-centric attribute-based access control. In: MMM-ACNS Proceedings, pp. 84–96 (2012)
21. Kafura, D.G., Gracanin, D.: An information flow control meta-model. In: Conti, M., Vaidya, J., Schaad, A. (eds.) 18th ACM Symposium on Access Control Models and Technologies, SACMAT, pp. 101–112. ACM (2013)
22. Kuhn, D.R.: Role based access control on MLS systems without kernel changes. In: Proceedings of the 3rd ACM Workshop on RBAC, pp. 25–32 (1998)
23. Kuhn, D.R., Coyne, E.J., Weil, T.R.: Adding attributes to role-based access control. IEEE Comput. **43**(6), 79–81 (2010)
24. Kumar, N.V.N., Shyamasundar, R.K.: A complete generative label model for lattice-based access control models. In: SEFM, Proceedings, pp. 35–53 (2017)
25. Mitra, B., Sural, S., Vaidya, J., Atluri, V.: Migrating from RBAC to temporal RBAC. IET Inf. Secur. **11**(5), 294–300 (2017)
26. Osborn, S.L.: Mandatory access control and role-based access control revisited. In: Proceedings of the 2nd Workshop on RBAC, pp. 31–40 (1997)
27. Osborn, S.L.: Information flow analysis of an RBAC system. In: SACMAT Proceedings, pp. 163–168 (2002)
28. Phillips, C., Demurjian, S., Ting, T.: Towards information assurance in dynamic coalitions. IEEE IAW, USMA (2002)
29. pyABAC: Attribute Based Access Control (ABAC) for python. https://py-abac.readthedocs.io. Accessed Dec 2020
30. Rajpoot, Q.M., Jensen, C.D., Krishnan, R.: Attributes enhanced role-based access control model. In: TrustBus Proceedings, pp. 3–17 (2015)
31. Sandhu, R.S.: Role hierarchies and constraints for lattice-based access controls. In: Computer Security - ESORICS 96, Proceedings, pp. 65–79 (1996)
32. Sandhu, R.S., Ferraiolo, D.F., Kuhn, D.R.: The NIST model for role-based access control: towards a unified standard. In: Fifth ACM Workshop on RBAC, Berlin, Germany, 26–27 July 2000, pp. 47–63 (2000)
33. Stambouli, A., Logrippo, L.: Data flow analysis from capability lists, with application to RBAC. Inf. Process. Lett. **141**, 30–40 (2019)
34. Tuval, N., Gudes, E.: Resolving information flow conflicts in RBAC systems. In: DBSec Proceedings, pp. 148–162 (2006)