Attribute Transformation for Attribute-Based Access Control

Prosunjit Biswas
Dept. of Computer Science &
Institute for Cyber Security
Univ. of Texas at San Antonio
prosun.csedu@gmail.com

Ravi Sandhu
Dept. of Computer Science &
Institute for Cyber Security
Univ. of Texas at San Antonio
ravi.sandhu@utsa.edu

Ram Krishnan
Dept. of Computer Science &
Institute for Cyber Security
Univ. of Texas at San Antonio
ram.krishnan@utsa.edu

ABSTRACT

In this paper, we introduce the concept of transforming attribute-value assignments from one set to another set. We specify two types of transformations—attribute reduction and attribute expansion. We distinguish policy attributes from non-policy attributes in that policy attributes are used in authorization policies whereas the latter are not. Attribute reduction is a process of contracting a large set of assignments of non-policy attributes into a possibly smaller set of policy attribute-value assignments. This process is useful for abstracting attributes that are too specific for particular types of objects or users, designing modular authorization policies, and modeling hierarchical policies. On the other hand, attribute expansion is a process of performing a large set of attribute-value assignments to users or objects from a possibly smaller set of assignments. We define a language for specifying mapping for the transformation process. We also identify and discuss various issues that stem from the transformation process.

Keywords

Attribute-Based Access Control, ABAC, Attribute Transformation, Attribute Reduction, Attribute Expansion

1. INTRODUCTION

Access control has been a major component in enforcing security and privacy requirements of information and resources with respect to unauthorized access. While many access control models have been proposed only three, viz., DAC, MAC and RBAC, have received meaningful practical deployment. DAC (Discretionary Access Control) [22] allows resource owners to retain control on their resources by specifying who can or cannot access certain resources. To address inherent limitations of DAC such as Trojan Horses, MAC (Mandatory Access Control) [22] has been proposed which mandates access to resources by pre-specified system policies. These models emphasize the specific policies of owner-based discretion for DAC and one-directional information flow in a lattice of security labels for MAC. On the other hand, RBAC (Role Based Access Control) [21] is a policy neutral, flexible and administrative-friendly model. Notably RBAC is capable of enforcing both DAC and MAC. MAC is also commonly referred to as LBAC (Lattice-Based Access Control).

Attribute Based Access Control (ABAC) has gained considerable attention from businesses, academia and standard bodies in recent years [11]. ABAC uses attributes on users, objects and possibly other entities (e.g. context, environment, actions) and specifies rules using these attributes to assert who can have which access permissions (e.g. read, write) on which objects.

Several attribute based access control models have been proposed in the literature [5, 15, 23, 27]. These models have been proposed for different application domains including Cloud Infrastructure [6, 3, 14, 20, 25], Grid Computing [17], Health Care [18], Internet of Things [2] and so on [7]. These domains typically include very large number of miscellaneous objects and many types of users. For example, a Cloud Infrastructure has many instances of Virtual Machines, Block Storage resources, Object Storage resources (eg. objects, containers, accounts), network resources (eg. firewalls, routers) and so on. All these objects have many attributes of their own. As a result there would be a large number of attributes, many of which are specific to particular types of objects and not meaningful to other object types. Furthermore, new attributes would be added as we add new object types in the system. Thus, similar to explosion of roles [8], there may be an *explosion of attributes*.

Having a large set of attributes incurs administrative difficulties from different perspectives for ABAC adoption. Firstly, it takes considerable effort to assign or de-assign these attribute values to users or objects. Secondly, authorization policies defined with these attributes would be large and complex in nature with resulting difficulty in specification, update, modification, review and so on.

In this paper, we introduce the concept of *attribute transformation* for converting one set of attribute-value assignments into another set of assignments. We specify two types of transformation—*attribute reduction* and *attribute expansion*. In the reduction process, we transform a large set of assignments into possibly a smaller set of assignments. This is useful for abstracting attributes that are too specific for particular type of objects or users, designing modular authorization policies, and modeling hierarchical policies. On the other hand, attribute expansion is the process of assigning

ABAC'17, March 24, 2017, Scottsdale, AZ, USA.
 ISBN 978-1-4503-4910-9/17/03…15.00
DOI: http://dx.doi.org/10.1145/3041048.3041052

larger set of attributes to users or objects from possibly a smaller set of assignments.

The motivation for attribute transformation is there in the literature. For example, attribute-based user-role assignment [1] reduces a set of user-attributes into user roles. Similarly, Kuhn, Coyne and Weil [1, 16] propose the concept of dynamic roles that uses attribute-based rules to derive user roles. These models only consider reduction of user attributes with respect to RBAC. We generalize this concept for both user and object attributes and present this notion for ABAC. On the other hand, HGABAC [23] presents an approach where users and objects may obtain multiple attribute-value assignments by respectively joining hierarchically related user and object groups.

We make the following contributions in the paper.

- We define a concept for transforming one set of attribute-value assignments into another set of assignments. We discuss two interesting transformation cases namely reduction and expansion. We delineate usefulness of these approaches for specification of authorization policies and larger set of attribute-value assignments.
- We define a language for specifying transformations.
- We discuss related issues emerging from reduction and expansion of attributes.

Rest of the paper is organized as follows. Section 2 presents related works. In Section 3, we define attribute transformation. Section 4 presents the concept of attribute reduction, its motivation, language for specifying reduction mapping along with issues emerging from the reduction process. Section 5 is about expansion of attributes including issues resulting from it. We conclude the paper in Section 6.

2. RELATED WORK

$ABAC_\alpha$[15] is among the first few models to formally define an ABAC model. It is designed to demonstrate flexibilities of an ABAC system to configure DAC, MAC and RBAC models. $ABAC_\alpha$ uses subset of subject attributes and object attributes to define an authorization policy for a particular permission p. It describes a constraint language to specify subject attributes from user attributes. Furthermore, it also presents a constraint language for changing object attributes at creation or modification time.

HGABAC [23] is another notable work in designing a formal model for an ABAC system. Besides designing a flexible policy language capable of configuring DAC, MAC and RBAC, it also addresses the problem of assigning attributes to a large set of users and objects. It specifies hierarchical groups and provides a mechanism for inheriting attributes from a group by joining to the group.

ABAC-for-web-services [27] is among very few earlier works to outline authorization architecture and policy formulation for an ABAC system. The authors propose a distributed architecture for authoring, administering, implementing and enforcing an ABAC system, albeit in a semi-formal formal language, including composing hierarchical policies from individual policies.

Wang et al [26] present a stratified logic programming based framework to specify ABAC policies. Even though, they only consider user attributes, they focus on providing a consistent, high performance and workable solution for ABAC system.

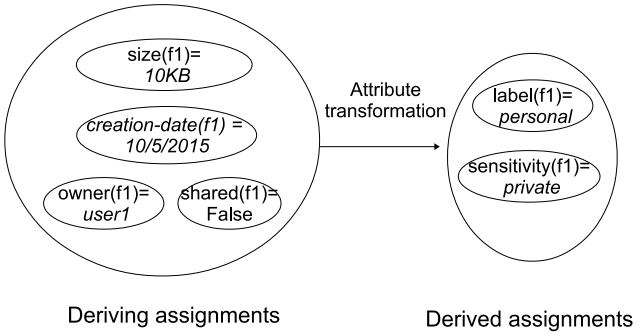

Figure 1: Example of attribute transformation

In its ABAC guide [11] and other publications [12], NIST defines common terminologies, and concepts for an ABAC system. The guide discusses required components, considerations and architecture for designing an enterprise ABAC system. It acknowledges the fact that ABAC rules can be quite complex in boolean combination of attributes or in simple relations involving attributes. Additionally, it discusses more advanced features like attribute and policy engineering, federation of attributes and so on. Nonetheless, these documents are focused towards establishing general definitions and considerations of an ABAC system without providing a concrete model definition.

Policy Machine [9] is another example of general purpose ABAC model. Instead of following the conventional approach of modeling attributes as functions, they model attributes as groups (eg. user groups, object groups etc). They also present the concept of *Policy Class* which provides the expressive power and flexibility for combining multiple security-policies easily [19].

There are other works that design an ABAC system from a particular application context. For example, WS-ABAC [24] is motivated by requirements in web services, ABAC-in-grid [17] is motivated by needs in the grid computing.

Another interesting line of work combines attributes with Role Based Access Control. Kuhn et. al [16] provides a framework for combining roles and attributes. In the framework, they briefly outline three different approaches—(i) dynamic roles which retain basic structure or RBAC and uses attribute based rules to derive user roles, (ii) attribute centric, which treat role as another ordinary attribute, and (iii) role centric, which uses roles to grant permissions and attributes to reduce permissions to be available to the user. Various other earlier or subsequent works involving roles and attributes can also be cast in Kuhn's framework.

There have been few works in administration of ABAC models. Most of them consider the problem of assigning attributes to users or objects. Examples include [10, 13].

3. ATTRIBUTE TRANSFORMATION

Attribute transformation is the process of transforming one set of attribute-value assignments into another set of assignments. For example, there might be attributes inherent to objects that we want to convert into different set of attributes that we can use in our authorization system. Figure 1 shows an example of attribute transformation. In the figure, the set of attributes used to derive the transformation are *size(), creation-date(), owner()* and *shared()*. These attributes are intrinsic to file objects in a system. On the

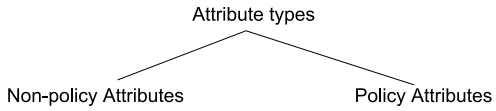

Figure 2: Attribute types

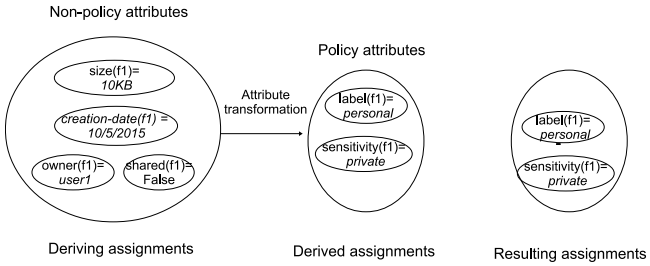

Figure 3: Attribute transformation (reduction)

other hand, the set of attributes that we derive through the transformation process are *label()* and *sensitivity()* which are security-related attributes and can be used to derive authorization decisions. We call the former set of attribute-value assignments that derives the transformation *deriving assignments* and the later set of assignments *derived assignments*. They are highlighted in Figure 1.

The motivation for attribute transformation includes the following.

Derive Policy Attributes. There are lots of attributes associated with users or objects. Examples include *size()*, *creation-date(), is-shared()* etc for files stored in a file system, *title(), author(),publisher()* etc for books, *age(), sex(), location(), role()* etc for users and so on. Many of these attributes are not useful for defining authorization policies. Many of the attributes that are useful are only meaningful to certain types of users or objects. For example, the attribute *publisher()* may be useful for books but not for files. As a result, defining authorization policies with these attributes is not practical because they would become unmanageable very soon. We envision, there would be a smaller and manageable set of attributes that we can use in defining authorization policies. This manageable set of attributes is called *Policy Attributes*. Figure 2 shows classification of all attributes into these two broad classes—*Policy Attributes* and *Non-policy Attributes*. To state informally *Policy Attributes* are the set of attributes that we use in defining authorization policies. We consider all other attributes as *Non-policy Attributes*.

Derive Attribute-value assignments. Even so there could be a large number of *Policy Attributes*. As a result, it would be difficult to manually assign values of all these

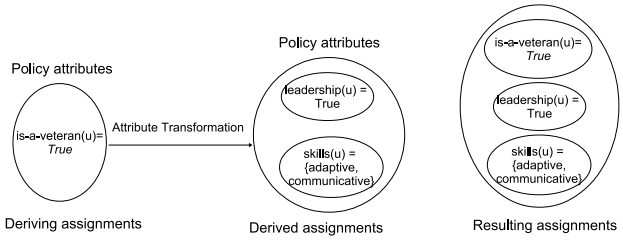

Figure 4: Attribute transformation (expansion)

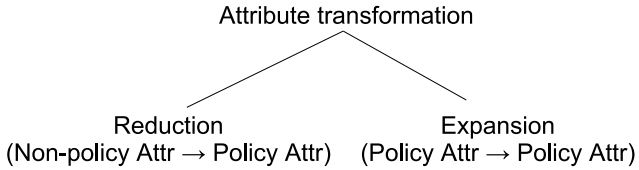

Figure 5: Transformation types

attributes. Moreover, assignment to some attribute-values naturally implies assignments to some other attribute-values. For example, someone having an assignment *is-veteran(u)= true*, may also imply following assignments—*leadership(u)= true* and *skill(u) = {adaptive, communicative}*.

We consider two examples of attribute transformation given in Figure 3 and Figure 4. In Figure 3, we transform Non-policy attributes into Policy attributes. As we only consider Policy Attributes to be used further, in this case, the resulting assignments are equivalent to the derived assignments. On the other hand, in Figure 4, we derive Policy Attribute from other Policy Attributes. As both deriving and derived assignments include Policy Attributes, we include both deriving and derived assignments in the resulting assignments.

Based on how to transform attributes, we classify *Attribute Transformation* into *Attribute Reduction* and *Attribute Expansion*. In Attribute Reduction, we transform Non-policy Attribute-value assignments into Policy Attribute-value assignments. In attribute-expansion, we transform assignments from Policy Attribute-values to Policy Attribute-values. Figure 5 shows this classification.

4. ATTRIBUTE REDUCTION

Attribute reduction is the process of transforming *Non-policy Attributes* into *Policy Attributes*. The motivation for reduction lies in the fact that there are lots of *Non-policy Attributes* that are hard to manage. For example, defining authorization policies with Non-policy Attributes is not pragmatic because it would soon inflate the size and complexity of authorization policies beyond manageable limit. For an example of attribute reduction, consider the mapping *sensitive-vm-mapping* ≡*resource-type(o)=VM* ∧ *image-type(o) = corporate* → *security-label(o) = sensitive*. It maps values from Non-policy Attributes named *resource-type* and *image-type* into a value of a single Policy Attribute named *security-label*. Benefits of attribute reduction includes the following.

Abstraction. Attribute reduction abstracts Non-policy Attributes and translates them to Policy Attributes meaningful to the security administrators. Consider the mapping *sensitive-firewall-mapping* ≡ *resource-type(o) = Firewall* ∧ *protocol(o) = UDP* ∧ *network(o) = Internal* → *security-label(o) = sensitive*. In this case, resource specific Non-policy Attributes like *resource-type*, *protocol*, *network* are mapped to the Policy Attribute *security-label*. A security architect may define authorization policies using *security-label* attribute values without having in depth knowledge of Virtual Machines or Firewalls. The mapping-rules like *sensitive-vm-mapping* or *sensitive-firewall-mapping* can be defined independently by security experts having proper domain knowledge.

Modular design. Let us consider the authorization policy *read-policy1* ≡ *security-label(o) = sensitive* ∧ *role(u)= manager* as shown in Figure 6. It authorizes managers to

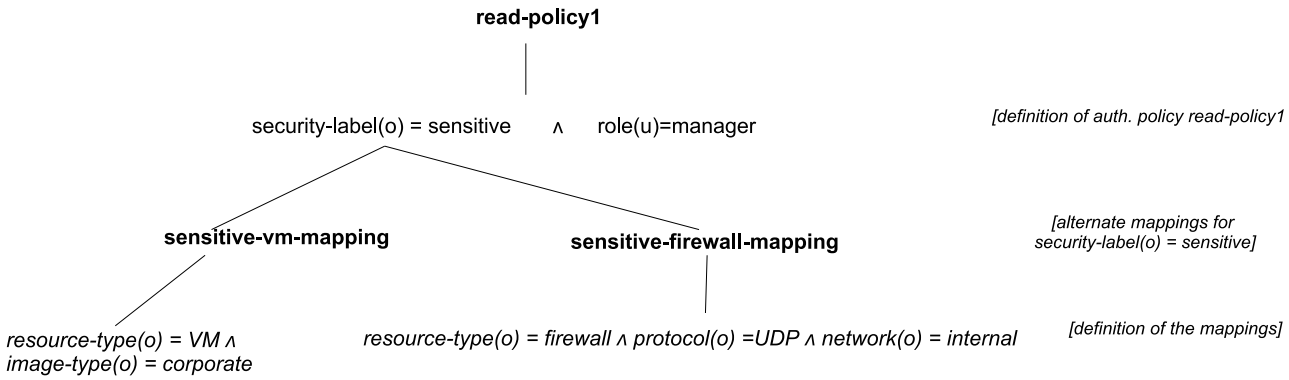

Figure 6: Example of an authorization policy fitting in reduction mappings

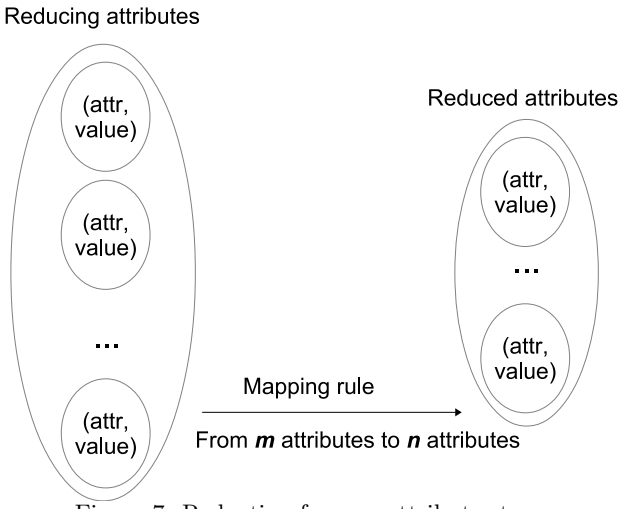

Figure 7: Reduction from m attributes to n attributes

read sensitive objects without explicitly specifying *sensitive* objects. Now *read-policy1* can easily be extended for Virtual Machine and Firewall resources by including the mappings *sensitive-vm-mapping* and *sensitive-firewall-mapping*. This facilitates modular as well as extendable designs.

Hierarchical policy. Attribute reduction results in hierarchies between authorization policies and mappings. For example, in Figure 6, *read-policy1* subsumes both sub-policies represented by mappings *sensitive-vm-mapping* and *sensitive-firewall-mapping*. An implication of hierarchical policies is that policies may be defined and enforced for or from multiple administrative scopes. For example, *read-policy1* can be defined for the whole organization whereas *sensitive-vm-mapping* and *sensitive-firewall-mapping* can be defined for and from the cloud infrastructure.

In the rest of this section, we first discuss some types and properties of mapping-based reduction. We then present a language for specifying reductions. Finally, we discuss some issues and complexities regarding attribute reduction.

4.1 Scopes and types of reduction

Attribute reduction can be applied to both users and objects. The motivation for user attribute reduction is there in the literature. For example, Mohammad and Sandhu [1] and Kuhn et. al [16] derive user roles from user attributes. We believe, there is even greater motivation for reducing object attributes. This is because, today's large protection system manages millions of resources/objects having hundreds or even thousands of resource-types. These resource-types have many low-level Non-policy Attributes meaningful to particular types. Moreover, new resource types may be added in the system which further increases the number of attributes.

Based on cardinality, we categorize reduction into three types—many-to-many, many-to-one and one-to-one. In many-to-many reduction, m number of attributes ($m > 1$) are reduced to n number of attributes ($1 < n < m$). These types of reduction are depicted in Figure 7. In many-to-one reduction, m attributes ($m > 1$) are reduced to a single attribute ($n = 1$). The *sensitive-vm-mapping* or *sensitive-firewall-mapping* shown in Figure 6 are examples of many-to-one reductions. In one-to-one reduction, one attribute is transformed into another attribute ($m = n = 1$) of different attribute-value ranges. A mapping from $age : U \rightarrow \{1, 2, ...100\}$ to $age\_group : U \rightarrow \{junior, adult, senior\}$ is an example of one-to-one mapping.

4.2 Mapping rules

The mapping rules for transforming attribute-value assignments are specified in Table 1. The rules are presented in BNF notation. Item I, II and III in Table 1 specify the terminal symbols, the non-terminal symbols and the start symbol respectively for the production rules.

The production rules are described in Item IV of the table. We specify two different production rules—one for transforming object attribute-value assignments and the other for transforming user attribute-value assignments. The rule *ObjAttrValAssgn* $\rightarrow$ *ObjAttrValAssgn* is interpreted in a way that existing object attribute-value assignments that satisfy the left-hand side of the rule implies assignments that are specified in the right-hand side of the rule. In the grammar, the symbol ObjAttrValAssgn is reduced to object attribute-value expression which is further reduced to combinations of object attribute-value pairs. We specify following set of object attributes $\{oa_1, oa_2, ..., oa_k\}$ and attribute-values are specified from the set $\{oav_1, oav_2, ..., oav_l\}$. For an example, the following rule *resource-type()=VM* $\wedge$ *image-type()* = *corporate* $\rightarrow$ *security-label()* = *sensitive* implies that the objects that are assigned *resource-type(o)* = *VM* and *image-type(o)* = *corporate*, are automatically assigned *security-label(o)* = *sensitive*. The other rule *UsrAttrValAssgn* $\rightarrow$ *UsrAttrValAssgn* is also interpreted in a similar way with

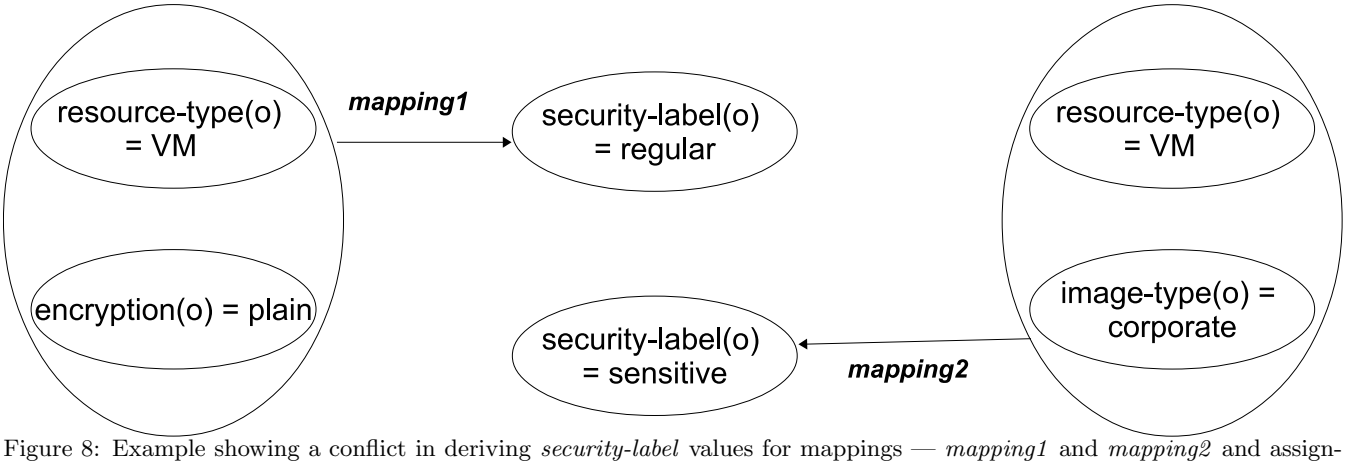

Figure 8: Example showing a conflict in deriving *security-label* values for mappings — *mapping1* and *mapping2* and assignments — *resource-type(o) = VM*, *image-type(o) = corporate* and *encryption(o) = plain*.

Table 1: Mapping rules

I. The terminal symbols

$\wedge$, $=$, $\rightarrow$,
oa_1, oa_2, ..., oa_k,
oav_1, oav_2, ..., oav_l,
ua_1, ua_2, ..., ua_m,
uav_1, uav_2, ..., uav_n

II. The non-terminal symbols

ObjAttrValAssgn, UsrAttrValAssgn,
ObjAttrValExpr, UsrAttrValExpr,
ObjAttrValPair, UsrAttrValPair,
ObjAttr, UsrAttr,
UsrAttrValue, ObjAttrValue

III. The start symbol

MappingRule

IV. The production rules (in BNF notation)

MappingRule::=
ObjAttrValAssgn $\rightarrow$ ObjAttrValAssgn |
UsrAttrValAssgn $\rightarrow$ UsrAttrValAssgn
ObjAttrValAssgn :: = ObjAttrValExpr
UsrAttrValAssgn :: = UsrAttrValExpr
ObjAttrValExpr ::= ObjAttrValPair |
ObjAttrValExpr $\wedge$ ObjAttrValExpr
UsrAttrValExpr ::= UsrAttrValPair |
UsrAttrValExpr $\wedge$ UsrAttrValExpr
ObjAttrValPair ::= ObjAttr = ObjAttrValue
UsrAttrValPair ::= UsrAttr = UsrAttrValue
ObjAttr ::= oa_1| oa_2|...| oa_k
ObjAttrValue ::= oav_1| oav_2|...|oav_l
UsrAttr ::= ua_1| ua_2|...| ua_m
UsrAttrValue ::= uav_1| uav_2|...|uav_n

user attributes {ua_1, ua_2,..., ua_m} and user attribute-values {uav_1, uav_2,..., uav_n}.

The rules presented above for attribute transformation are representative. We keep it simple to convey the idea succinctly. In attribute-value expression (eg. ObjAttrValAssgn), we use only conjunction operator ($\wedge$). Disjunction of assignments can be captured by defining multiple mapping rules. While comparing attribute values, we only match for equality. More generally, we could allow comparisons such as $>$ or $\geq$.

4.3 Issues in reduction

In this section, we discuss some issues arising from attribute reduction.

4.3.1 Conflicts resulting from multiple mappings

Here we discuss conflicts that might occur in presence of multiple mappings and given assignments. For example, consider *mapping1* and *mapping2* and attribute-value assignments—*resource-type(o) = VM*, *image-type(o) = corporate* and *encryption(o) = plain* as shown in Figure 8. Given these assignments, *mapping1* derives *security-label* value as *regular* and *mapping2* derives *security-label* value as *sensitive*. This might be undesirable and/or conflicting in a protection system. We specify some approaches to be used altogether or exclusively to mitigate these conflicts.

Resolve conflicts. There are different ways to resolve conflicts. We can set distinct priority with each derived attribute values. For example, we can set a priority such that the attribute-value *sensitive* overrides all other attribute-values for the attribute *security-label*. Another approach can be to accept both values for a set-valued attribute.

Avoid conflicts. As opposed to resolving conflicts, we might try safe assignments such that no conflict might occur. In the context of Figure 8, we may regulate assignments so that a virtual machine that uses corporate images never remain unencrypted. Constraint free assignments of attributes are treated in detail by Bijon, Krishnan and Sandhu [4].

4.3.2 Assigned vs derived attribute-value

Values of a Policy Attribute can be explicitly assigned by administrators or be derived using reduction mappings. Figure 9 presents such a scenario. In Figure 9, the assigned value for *security-label* attribute is *sensitive* whereas the de-

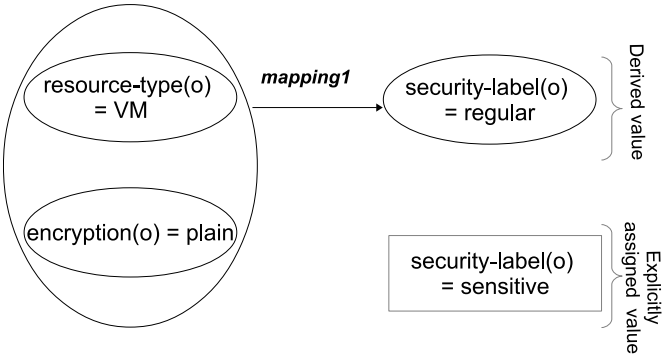

Figure 9: Conflict in derived and assigned attribute-value

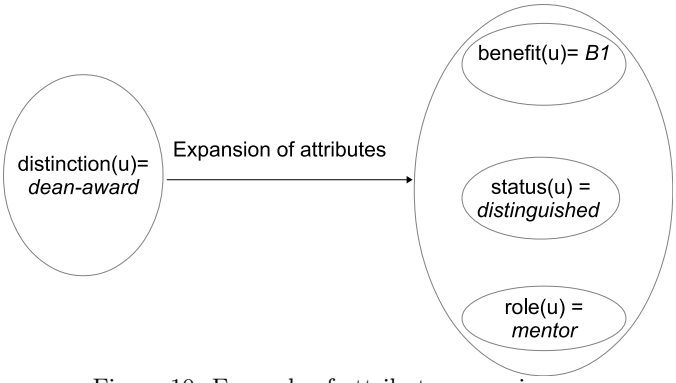

Figure 10: Example of attribute expansion

rived value is *regular*. Different strategies may be applied for resolving this type of conflict. One simple strategy is that assigned values override derived values (or vice versa). Another strategy can be attributes whose values are derived are never assigned (or vice versa).

5. ATTRIBUTE EXPANSION

Attribute expansion is the process of assigning a large set of *Policy Attribute* values from a given, possibly smaller set of *Policy Attribute-value* assignments. For example, consider the scenario given in Figure 10 where as part of receiving an annual award *dean-award*, a student automatically receives following assignments—*benefit(u)=B1*, *status(u)=distinguished*, and *role(u)=mentor*. The advantage of this form of assignments is that it derives additional assignments saving manual administrative efforts for the same purpose. As long as the award does not expire, the student needs all the assignments to fulfill his responsibilities properly but after expiration the student is required to be de-assigned appropriately. In the context of this example, we assume all these attributes are *Policy Attributes*, so that both deriving and derived assignments remain as resulting assignments.

The motivation of attribute expansion is there in the literature. For example, HGABAC [23] presents an approach where users/objects may obtain multiple attribute-value assignments as part of joining user/object groups. HGABAC also uses the concept of hierarchical groups to obtain multiple/cascading assignments. We generalize these concepts using attributes and avoiding non-attribute entities in an ABAC system.

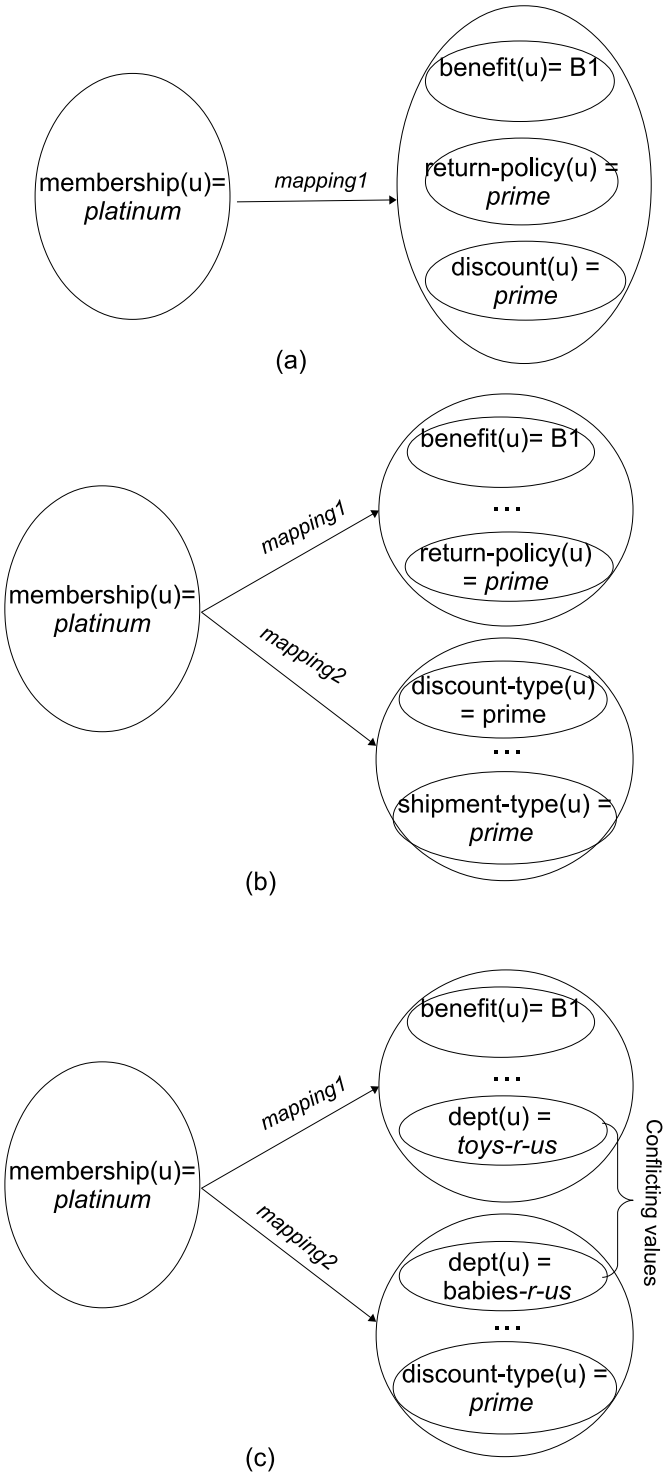

Figure 11: Attribute expansion in presence of (a). one mapping-rule, (b). multiple non-conflicting rules, and (c). multiple conflicting rules

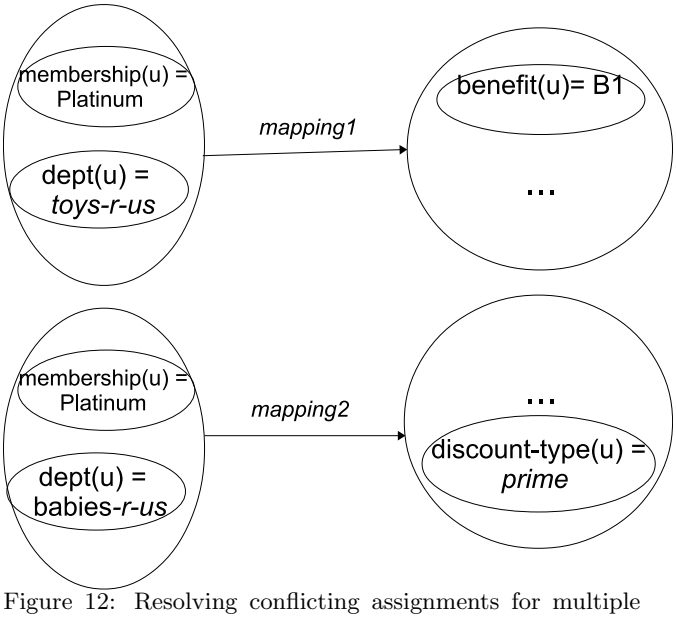

Figure 12: Resolving conflicting assignments for multiple mapping-rules (for cases in Figure 11(c))

5.1 Mapping rules

As both expansion and reduction are transformation of attributes, we use reduction rules as described in Section 4.2 for expansion mappings.

5.2 Issues in expansion

We identify three different cases for expanding attributes based on how multiple mappings interact. We illustrate both valid and invalid cases with examples in Figure 11. In Figure 11(a), there is only one mapping from *membership(u) = platinum* to *benefit(u) = B1*, *return-policy(u) = prime* and *discount(u) = prime*. This case is inherently conflict free. Figure 11(b) presents a case where there are multiple mappings for *membership(u)=platinum*. As we see, these mappings assign values to disjoint set of attributes. As a result, these mapping are conflict free as well. In Figure 11(c), *mapping1* sets *dept(u)=toys-r-us* where as *mapping2* sets *dept(u)=babies-r-us* which is a conflict. We can adopt mitigation approaches similar to that of attribute reduction presented in Section 4.3.1. Another approach can be to remove the conflicting assignments from the right-hand side of the mapping-rules and add them to the left-hand side of the same rule. For example, for the conflicting assignment of Figure 11(c), we can remove the assignment *dept(u)* from the right-hand side and add it to the left-hand side as shown in Figure 12.

6. CONCLUSION & LIMITATION

We introduce the concept of attribute transformation for Attribute-Based Access Control. We describe two different types of transformation—*Attribute Reduction* and *Attribute Expansion*. Attribute reduction is a process of converting a large set of attribute-value assignments into possibly smaller set of assignments. Advantages of attribute reduction include abstraction of attributes, modular policy design, hierarchical policy etc. Attribute expansion, which converts a smaller set of assignments into a larger set, is useful for assigning large number of attribute-values to users or objects. We describe a mapping-language for the transformation process. For each type of transformation, we identify cases which result conflicts and provides alternates to resolve them.

The reduction process we specify here is useful for defining new authorization policies. It would be interesting to investigate how to redesign existing authorization policies using given reduction rules. Moreover, we do not provide any guideline to evaluate reduction mapping as there might be more than one such equivalent mappings. We briefly describe how reduction mapping may result hierarchy between policies. Further investigation in this direction is required.

Acknowledgments

This research is partially supported by NSF Grants CNS-1111925 and CNS-1423481, and DoD ARL Grant W911NF-15-1-0518.

7. REFERENCES

[1] M. Al-Kahtani and R. Sandhu. A model for attribute-based user-role assignment. In *18th Annual Proceedings of Computer Security Applications Conference*, pages 353–362. IEEE, 2002.

[2] A. Alshehri and R. Sandhu. Access control models for cloud-enabled internet of things: A proposed architecture and research agenda. In *Proceedings of the Workshop on Privacy in Collaborative & Social Computing*, 2016.

[3] S. Bhatt, F. Patwa, and R. Sandhu. An attribute-based access control extension for openstack and its enforcement utilizing the policy machine. In *International Conference on Network and System Security*, 2016.

[4] K. Bijon, R. Krishnan, and R. Sandhu. Mitigating multi-tenancy risks in IaaS cloud through constraints-driven virtual resource scheduling. In *Proc. of the 20th Symposium on Access Control Models and Technologies*, pages 63–74. ACM, 2015.

[5] P. Biswas, R. Krishnan, and R. Sandhu. Label-based access control: an ABAC model with enumerated authorization policy. In *Proc. of ABAC*, pages 1–12. ACM, 2016.

[6] P. Biswas, F. Patwa, and R. Sandhu. Content level access control for Openstack Swift storage. In *Proceedings of the 5th Conference on Data and Application Security and Privacy*, pages 123–126. ACM, 2015.

[7] P. Biswas, R. Sandhu, and R. Krishnan. An attribute-based protection model for JSON documents. In *International Conference on Network and System Security*, pages 303–317. Springer, 2016.

[8] A. Elliott and S. Knight. Role explosion: Acknowledging the problem. In *Software Engineering Research and Practice*, pages 349–355, 2010.

[9] D. Ferraiolo, V. Atluri, and S. Gavrila. The Policy Machine: A novel architecture and framework for access control policy specification and enforcement. *Journal of Systems Architecture*, 57(4):412–424, 2011.

[10] M. Gupta and R. Sandhu. The GURA$_G$ administrative model for user and group attribute assignment. In *Int. Conference on Network and Sys. Security*, pages 318–332. Springer, 2016.

[11] V. C. Hu, D. Ferraiolo, R. Kuhn, A. Schnitzer, K. Sandlin, R. Miller, and K. Scarfone. Guide to attribute based access control (ABAC) definition and considerations. *NIST Special Publication*, 800:162, 2014.

[12] V. C. Hu, D. R. Kuhn, and D. F. Ferraiolo. Attribute-based access control. *IEEE Computer*, (2):85–88, 2015.

[13] X. Jin, R. Krishnan, and R. Sandhu. A role-based administration model for attributes. In *Proceedings of the 1st International Workshop on Secure and Resilient Architectures and Systems*, pages 7–12. ACM, 2012.

[14] X. Jin, R. Krishnan, and R. Sandhu. Role and attribute based collaborative administration of intra-tenant cloud iaas. In *International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, pages 261–274. IEEE, 2014.

[15] X. Jin, R. Krishnan, and R. S. Sandhu. A unified attribute-based access control model covering DAC, MAC and RBAC. *DBSec*, 12:41–55, 2012.

[16] D. R. Kuhn, E. J. Coyne, and T. R. Weil. Adding attributes to role-based access control. *IEEE Computer*, (6):79–81, 2010.

[17] B. Lang, I. Foster, F. Siebenlist, R. Ananthakrishnan, and T. Freeman. A flexible attribute based access control method for grid computing. *Journal of Grid Computing*, 7(2):169–180, 2009.

[18] M. Li, S. Yu, Y. Zheng, K. Ren, and W. Lou. Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption. *IEEE Transactions on Parallel and Distributed Systems*, 24(1):131–143, 2013.

[19] P. Mell, J. M. Shook, and S. Gavrila. Restricting insider access through efficient implementation of multi-policy access control systems. In *Proceedings of the International Workshop on Managing Insider Security Threats*, pages 13–22. ACM, 2016.

[20] C. Ngo, Y. Demchenko, and C. de Laat. Multi-tenant attribute-based access control for cloud infrastructure services. *Journal of Information Security and Applications*, 27:65–84, 2016.

[21] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, (2):38–47, 1996.

[22] R. S. Sandhu and P. Samarati. Access control: principle and practice. *IEEE Communications Magazine*, 32(9):40–48, 1994.

[23] D. Servos and S. L. Osborn. HGABAC: Towards a formal model of hierarchical attribute-based access control. In *Foundations and Practice of Security*, pages 187–204. Springer, 2014.

[24] H.-b. Shen and F. Hong. An attribute-based access control model for web services. In *PDCAT'06.*, pages 74–79. IEEE, 2006.

[25] Z. Wan, J. Liu, and R. H. Deng. HASBE: A hierarchical attribute-based solution for flexible and scalable access control in cloud computing. *IEEE Transactions on Information Forensics and Security*, 7(2):743–754, 2012.

[26] L. Wang, D. Wijesekera, and S. Jajodia. A logic-based framework for attribute based access control. In *Proceedings of FMSE '04*, pages 45–55. ACM, 2004.

[27] E. Yuan and J. Tong. Attributed based access control (ABAC) for web services. In *Proceedings of IEEE International Conference on Web Service*. IEEE, 2005.