A System for Centralized ABAC Policy Administration and Local ABAC Policy Decision and Enforcement in Host Systems using Access Control Lists*

David Ferraiolo
National Institute of Standards and Technology
Gaithersburg, Maryland 20899
USA
dferraiolo@nist.gov

Serban Gavrila
National Institute of Standards and Technology
Gaithersburg, Maryland 20899
USA
serban.gavrila@nist.gov

Gopi Katwala
National Institute of Standards and Technology
Gaithersburg, Maryland 20899
USA
gopi.katwala@nist.gov

ABSTRACT

We describe a method that centrally manages Attribute-Based Access Control (ABAC) policies and locally computes and enforces decisions regarding those policies for protection of resource repositories in host systems using their native Access Control List (ACL) mechanisms. The method is founded on the expression of an ABAC policy that conforms to the access control rules of an enterprise and leverages the ABAC policy expression by introducing representations of local host repositories into the ABAC policy expression as objects or object attributes. Repositories may be comprised of individual files, directories, or other resources that require protection. The method further maintains a correspondence between the ABAC representations and repositories in local host systems. The method also leverages an ability to conduct policy analytics in such a way as to formulate ACLs for those representations in accordance with the ABAC policy and create ACLs on repositories using the ACLs of their corresponding representations. As the ABAC policy configuration changes, the method updates the ACLs on affected representations and automatically updates corresponding ACLs on local repositories. Operationally, users attempt to access resources in local host systems, and the ABAC policy is enforced in those systems in terms of their native ACLs.

*The U.S. Government has filed a patent application of certain aspects of the subject matter disclosed in this paper.

ABAC'18, March 19–21, 2018, Tempe, AZ, USA
ACM 978-1-4503-5633-6/18/03
https://doi.org/10.1145/3180457.3180460

ACM Reference Format:
David Ferraiolo, Serban Gavrile and Gopi Katwala. 2018. A System for Centralized ABAC Policy Administration and Local ABAC Policy Decision and Enforcement in Host Systems using Access Control Lists. *In Proceedings of 3rd Workshop on Attribute Based Access Control (ABAC'18)*. ACM, Tempe, AZ, USA, 8 pages, March.
DOI: 10.1145/3180457.3180460

KEYWORDS

ABAC; Attribute Based Access Control; ACLs; Access Control Lists; NGAC; Next Generation Access Control; Architecture; Authorization; Policy Machine

1. INTRODUCTION AND BACKGROUND

We describe a method for the enforcement of Attribute Based Access Control (ABAC) policies in host systems using their Access Control Lists (ACLs). The method centrally manages ABAC policies using a Policy and Attribute Administrative Point and a database for storing attributes and policy information in what we refer to as the Minimum ABAC implementation.

The Minimum ABAC implementation also includes a Policy Analytics Engine, that computes Access Control Lists (ACLs) in terms of local host repositories that are represented as ABAC objects or object attributes. As a consequence of the method, ABAC policies are enforced over user access requests to resource repositories in local host systems in terms of their ACLs.

An ACL is a simple mechanism that dates back to the early 1970s and remains in widespread use to protect resource repositories of varying types (e.g., files and directories). Each resource repository is associated with an ACL that stores the users and their approved access rights for the repository. The list is checked by the access control system to determine if access is granted or denied. Lists need not be excessively long if groups of users with common access rights to the repository (rather than individual members) are attached.

The principal advantages of ACL mechanisms are that they are extremely efficient when computing access decisions and

help simplify the review of users' access rights to a repository. Another advantage is that they allow access to a repository to be easily revoked by simply deleting an ACL entry, or deleting a user or group membership of an ACL entry. However, because ACLs make it difficult to determine the access rights users have to repositories, ACLs are cumbersome when managing access capabilities of users.

Attribute-Based Access Control (ABAC) represents the latest milestone in the evolution of authorization approaches [1]. ABAC is an access control method wherein user requests to perform operations on resources are granted or denied based on the assigned attributes of the user, the assigned attributes of the resource, and a set of policies that are specified in terms of those attributes.

A key ABAC advantage is how easily it manages policy. When a user enters on duty or when a user's job function, authority, affiliations, or any other user characteristic changes, an administrator simply assigns/reassigns the user to the appropriate attributes, and the user automatically gains appropriate access capabilities to system resources. Similarly, when a resource is created or different accesses to a resource are required, appropriate object attribute assignments are created/deleted, automatically enabling policy-preserving user access rights to the resource.

ABAC implementations typically include four layers of functional decomposition working together to bring about policy-preserving access control: Enforcement, Decision, Access Control Data, and Administration. Among these components is a Policy Enforcement Point (PEP) that traps access requests and enforces policy. To determine whether to grant or deny access, the PEP submits the request to a Policy Decision Point (PDP). The PDP computes and returns a decision to the PEP based on policy and attribute information stored in one or more databases (access control data). Information is managed in the policy and attribute store through an ABAC system administrative API.

There are currently two standards [2] that address these ABAC features: Extensible Access Control Markup Language (XACML) [3] and Next Generation Access Control (NGAC) [4, 5, 6]. For both standards, there exist open-source and commercial-compliant implementations. While these implementations deliver ABAC's administrative advantages, not all ABAC implementations enable efficient policy analytics—the ability to answer key questions regarding the access state.

In many ways, the method offers the best of both ABAC and ACLs. By leveraging an ABAC implementation, the method provides a means of access control policy support that goes beyond what is feasible through direct management of ACLs. For instance, enforced policies may combine privileges of sub-policies (e.g., discretionary access control and role-based access control) and may consider denials for expressing privilege exceptions to sub-policies. By expressing policies in terms of combinations of user attributes, the method requires the creation and management of fewer attributes than the number of otherwise required groups. By conducting policy enforcement and decision-making using ACLs, the method provides enhanced performance in granting or denying user access requests beyond what is possible using an ABAC system alone. This enhanced performance is not only desirable in applications that manually access system resources on an individual basis, but is absolutely essential in such environments as big data processing and supercomputing, that require batch processing. Although ACLs preclude the ability to answer important policy review questions, the method allows the full breadth of policy analytics that is permissible to ABAC in general including the identification of the access capabilities of a user. Perhaps most appealing, the method achieves enforcement of ABAC policies in local host systems with minimal or no required changes to those systems beyond implementation of agent software.

2. MINIMUM ABAC REQUIREMENTS

The method of central management of ABAC policies and local enforcement of those ABAC policies through ACLs on local host repositories is dependent on an efficient means of conducting policy analytics in an ABAC system. Determining what group of users can access a resource with an access right (e.g., read or write) is especially crucial. The open source implementation, Harmonia 1.6, is an NGAC reference implementation on GitHub that exemplifies an ABAC implementation with the capability to efficiently conduct policy analytics [7]. Annex A of [6] describes, in detail, a linear-time algorithm to calculate the access rights a user has to objects representing protected resources based on the work published in [8]. The algorithm can also be easily adapted to make various other key policy determinations such as identifying the access rights a group of users have to protected resources.

Although a PEP and PDP are normally included in an ABAC implementation, the method does not depend on these components since the functions of enforcing ABAC policy and computing decisions are achieved by the local host's ACL mechanism. What is required of the ABAC system, in addition to conducting policy analytics, is the ability to administer and store policies and attributes. We generally refer to this administrative component as the Policy and Attribute Administrative Point, although XACML does not prescribe a means of managing its attributes.

3. METHOD

The method for centralized ABAC policy management and local host enforcement of ABAC policies using native host ACLs includes the following steps:

- Expressing an ABAC policy that defines privileges, or privileges and prohibitions, of users using a policy and attribute administration point for configuring the authorization data of a centralized ABAC system that, in part, defines policies in terms of objects and object attributes;
- Introducing representations of resource repositories needing protection in local systems that protect their data using ACLs into the ABAC policy expression as objects or object attributes;
- Establishing a one-to-one correspondence between the representations of resource repositories and actual resource repositories;
- Formulating ACLs for representations in accordance with the ABAC policy by determining the group of users that can exercise the access right for each access right (e.g., read, write) relevant for a representation r;
- Creating a group on the local system with user members for each determined group and hosting the resource repository corresponding to representation r, using agent software;

- Creating a user account, if one does not yet exist, for each user member of each created group on the local system hosting the resource repository corresponding to representation r, using agent software;
- Creating an ACL on the resource repository corresponding to representation r, using the ACL formulated for representation r and agent software;
- When required, altering the expression of ABAC policy using the policy and attribute administration point of the centralized ABAC system and mandating the update of ACLs of each representation affected by the alteration;
- Updating group memberships, user accounts, and ACLs on local systems with resource repositories that correspond to affected representations using agent software.

Although the method could be implemented in different ways, Figure 1 illustrates a preferred approach that includes a Control Center surrounded on three sides by an Administrator, a Minimum ABAC implementation, and a local Host System.

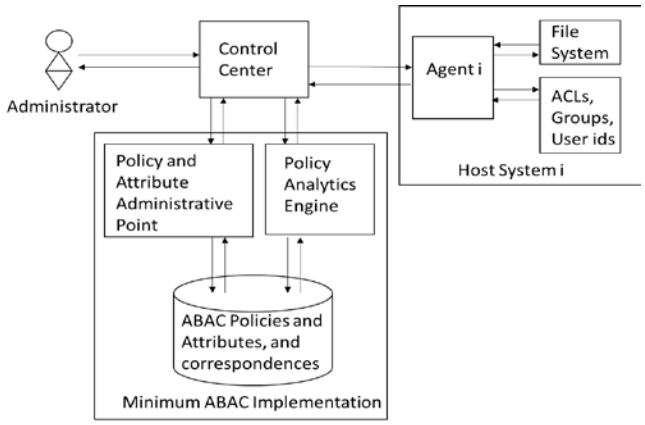

Figure 1: A preferred architecture of method

Administrators express ABAC policies, introduce representations of local repositories into the policy expression, and instruct the creation of ACLs for repositories as administrative commands using the administrative API of the ABAC Control Center. The Control Center provides status and resulting information in reply to administrative commands. The Minimum ABAC Implementation consists of a Policy and Attribute Administrative Point, a Policy Analytics Engine, and a database for storing ABAC Policies and Attributes. In addition, the method may store correspondences between repositories and representations in the database.

The Control Center, through the Policy and Attribute Administrative Point, creates and manages ABAC policies and attributes that are stored in computer memory and/or on disk referred to here as the database. The Control Center issues commands to the Policy and Attribute Administrative Point for managing attributes and policies. The Policy and Attribute Administrative Point implements administrative routines that, when executed, create and delete information stored in the database. These administrative routines may pertain to viewing or reading database information, which would be returned to the Control Center.

The Host System normally implements a File System comprised of repositories of files and directories and normally maintains an access control system with data comprising ACLs, groups, and user identities. In addition to these native components, the method implements Agent software on the Host System with administrative privileges for identifying and viewing repositories and creating, deleting, and updating groups, user identities, and ACLs for repositories. The main function of Agent software is to translate centralized Control Center administrative commands to native host administrative commands. Although the commands issued to Agent software by the Control Center may be uniform across a variety of Host Systems, Agent software on Host Systems are specific to the ACL, group, and user semantics of a host and, in this case, Host i. Agent software response to the Control Center may be uniform across Host Systems. Agent commands to the File System and commands to the host access control system are host-specific. Similarly, status and data returned to the Agent from the File System and access control system status information returned to the Agent are also host-specific.

The Control Center, through Agent software identifies repositories requiring protection in the File System, creates a representation of each such repository as either an object or an object attribute in the ABAC Policy using the Policy and Attribute Administrative Point, and creates a correspondence between the representation and repository in the database.

The Control Center, through the Policy Analytics Engine, computes ACLs with required groups for representations in accordance with ABAC Policies and Attributes stored in the database and subsequently creates ACLs for corresponding repositories, creates groups, and, if necessary, creates user identities in the host access control system using host agent software. To complete this function, the Control Center passes a representation (an object or object attribute) to the Policy Analytics Engine, which then issues read commands to the database resulting in the returns of requested ABAC policy and attribute data. Once the Policy Analytics Engine computes an ACL with required groups, that information is passed back to the Control Center.

The Control Center, through the Policy and Attribute Administrative Point, may update ABAC policies and/or attributes stored in the database. In such cases, the Control Center instructs the Policy Analytics Engine to re-compute ACLs and Groups for affected representations. Using Agent software, ACLs are updated for corresponding repositories, groups, and, if necessary, creates/deletes user identities in the host access control system.

The Policy and Attribute Administrative Point and Policy Analytics Engine could be built as modules of the Control Center on the same machine. The database could be hosted on that machine, or these components could reside as independent network components. Although portrayed as a single store, the attributes and policies may physically reside in different stores. In the case that the method provides ABAC support to a single host system, the Control Center, the entire Minimum ABAC Implementation, and the Agent could reside on that host system.

4. ILLUSTRATIVE EXAMPLE

Figure 2 illustrates an example directory structure of a file system on a host system owned by a Bank to serve as a running use case to highlight the features of the method. The structure includes a root directory ("Products") with two subdirectories ("loans" and "accounts"), each with subdirectories (e.g., loans 2 and accounts 1) for storing and organizing loan and account

products as files and with respect to the branches of the bank. For instance, loans 2 maintains loan files belonging to branch 2.

Although ACL features for protecting resource repositories can vary from system to system and different terminology is sometimes used to express the same feature, we identify semantics common to most if not all ACL mechanisms.

- ACLs on directories are treated differently than ACLs on files.
- Read on a directory implies the right to list children of the directory.
- Write on a directory implies the right to create/delete children of the directory.
- Read and write on a file implies the same right.
- ACLs on a directory or file can inherit or block ACLs of parent directories.

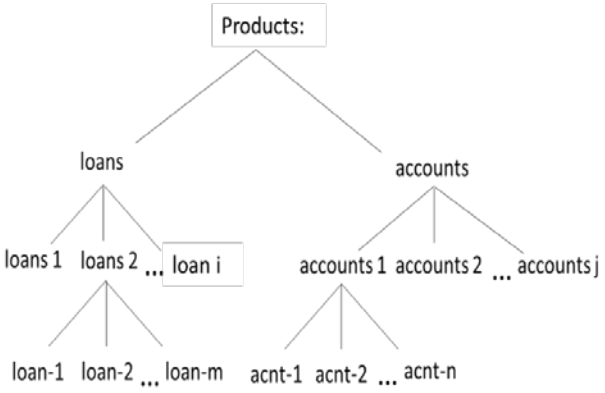

Figure 2: Example directory structure

In addition to the directory structure illustrated in Figure 2, we assume these ACL semantics for the purposes of our illustrative example.

4.1 ABAC Policy Expression

The method begins with the creation of an ABAC policy using the Policy and Attribute Administrative Point of an ABAC implementation. Figure 3 is an illustration of an example bank policy in terms of NGAC policy elements and relations wherein users (e.g., u1, u2) and user attributes (e.g., Teller, Branch1) are shown on the left side of the graph, and object attributes (e.g., Accounts 1 and Loans) and objects (e.g., **loan-1**, o2) are on the right side. The arrows denote assignments and imply a containment relation (e.g., loan-1 is contained in loans 2, Loans, Br2 Products, Products, and RBAC). The policy takes into consideration two sub-policies referred to by NGAC as policy classes: RBAC and BranchAccess.

Access rights to perform operations are acquired through associations. The dashed lines illustrate association relations. By *ua---ars---oa,* we denote an association where *ua* is a user attribute, *ars* is a set of resource and/or administrative access rights, and *oa* is an object attribute[1]. The ars depicted in Figure 3, pertain to both resource access rights and administrative access rights. The r and w are read and write, resource access rights, and c-ooa and d-ooa are administrative access rights for "creating an object in object attribute" and "deleting an object in object attribute." The meaning of an association *ua---ars---oa* is that the users contained in *ua* can execute the access rights in *ars* on the policy elements referenced by *oa*. The set of policy elements referenced by *oa* is dependent on (and meaningful to) the access rights in *ars*. For instance, the association Loan Officer---{r, w, c-ooa, d-ooa}---Loans pertains to capabilities to read and write objects (representing files) contained in Loans (i.e, o2 and loan-1) and create and delete object assignments (a type of relation) in Loans, Loans 1, and Loans 2.

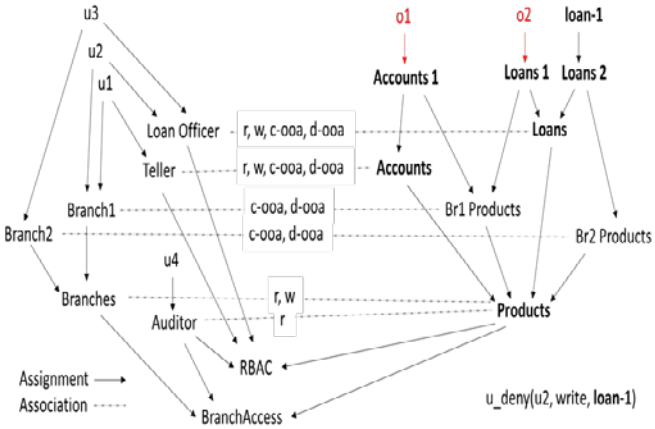

Figure 3: Example policy configuration

Collectively, associations and assignments indirectly specify privileges with respect to policy classes of the form (*u*, *ar*, *e*), with the meaning that user *u* is permitted (or has a capability) to execute the access right *ar* on element *e*, where *e* can represent an object attribute or object.

NGAC includes an algorithm for determining privileges with respect to one or more policy classes and associations. Specifically, (*u*, *ar*, *e*) is a privilege if and only if, for each policy class *pc* in which *e* is contained, the following is true:

1. The user *u* is contained by the user attribute of an association;
2. The element *e* is contained by the attribute *at* of that association;
3. The attribute *at* of that association is contained by the policy class *pc*; and
4. The access right *ar* is a member of the access right set of that association.

Table 1 lists the derived privileges for the policy configuration depicted in Figure 3.

Table 1. List of derived privileges for Figure 2

(u1, r, o1), (u1, w, o1), (u2, r, o2), (u2, w, o2), (u2, r, loan-1), (u2, w, loan-1), (u3, r, o2), (u3, w, o2), (u3, r, loan-1), (u3, w, loan-1), (u1, c-ooa, Accounts 1), (u1, d-ooa, Accounts 1), (u2, c-ooa, Loans 1), (u2, d-ooa, Loans 1), (u3, c-ooa, Loans 2), (u3, d-ooa, Loans 2), (u4, r, o1), (u4, r, o2), (u4, r, o3), (u4, r, loan-1)

[1] For the purposes of this paper, we specify an association using a simpler notation than formally specified in the NGAC standard.

In addition to assignments and associations, NGAC includes prohibitions or deny relations. In general, deny relations specify privilege exceptions. Among these prohibitions is a user-based deny, denote by, u_deny(*u*, *ars*, *pe*), where *u* is a user, *ars* is an access right set, and *pe* is a policy element used as a reference for itself and the policy elements contained by the policy element. The meaning is that user *u* cannot execute access rights in *ars* on policy elements in *pe*. User-deny relations can be created by an administrator. An administrator, for example, might impose a condition wherein no user is able to alter their own loan file, even if the user is assigned to Loan Officer with capabilities to read/write all Loans. The u-deny relation depicted in Figure 3, prohibits u2 from writing to **loan-1**. This privilege exception is reflected in Table 1 using red font.

A natural language description of the policy expressed by Figure 3 is as follows:

- Tellers can read and write accounts objects in all branches.
- Tellers can create and delete accounts objects in the branches for which they are assigned.
- Loan Officers can read and write loans objects in all branches.
- User u3 (a Loan Officer) cannot write to Loan-1.
- Loan Officers can create and delete loans objects in the branches to which they are assigned.
- An Auditor can read account and loan products in all branches.

4.2 Creating ACLs for Representations

The method leverages an ABAC policy expression by introducing representations of host repositories as either an object attribute in the case of a directory or an object in the case of a file. The method further maintains a correspondence between the ABAC representations of the repository and the actual repository in host systems. In Figure 3, **Accounts 1**, **Loans 1**, **Loans 2, Accounts, Loans, Products,** and **loan-1** are in bold to indicate that they represent host system repositories in the directory structure depicted in Figure 2.

Figure 4 illustrates an establishment of a correspondence between **Loans 2** in the ABAC configuration and loans 2 in the directory structure of the local host file system.

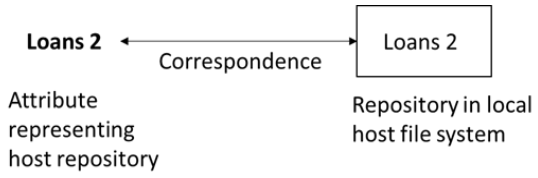

Figure 4: Correspondence between the representation of Loans 2 in the ABAC system and loans 2 in the local host File System.

Once a representation has been established, the method conducts a policy review in such a way as to formulate an ACL for the representation in accordance with the ABAC policy. A central aspect of the policy review involves determining the group of users who can perform specific operations (e.g., read and write) on the representation or on an object contained in the representation. Since the meaning of an ACL differs for directories and files, the logic of the Policy Analytics Engine may make a distinction between representations of files, directories containing files, and directories that do not contain files. For the purposes of this paper we assume a Policy Analytics Engine that makes such a distinction. In describing its logic, we use the notion of a "Custom" ACL to indicate the blocking of ACL privilege inheritance of parent directories.

Let us consider **loan-1**, a representation of the file loan-1. To read **loan-1** a user needs to be assigned to Loan Officer or Auditor. The group of users that meet this criterion are u2, u3, and u4. To write **loan-1** a user needs to be assigned to Loan Officer. The group of users that meet this criterion are u2 and u3. However, in accordance with the overall policy, u2 is denied the ability to write to **loan-1**, and, as such, user u2 is not included in the group for writing. Any convention can be used for naming groups. In our example, we will use gr1 for the group that can read and gr2 for the group that can write to **loan-1** in deriving an ACL for **loan-1**:

loan-1: Custom
gr1, r; gr2, w -- where gr1=u2, u3, u4, and gr2=u3

The ACL is designated as "Custom" to indicate that it does not inherit access rights from its parent directory (**loans 2**). In the case of a representation of a directory containing files, the logic creates a custom ACL for the directory and an ACL for inheritance by the files (the directory's children). While establishing correspondence with a directory repository that contains files, the logic also creates an arbitrary-unique object and assigns that object to the repositories representation if no object is currently assigned to the representation. The red object to object-attribute assignments in Figure 3 illustrates such an assignment. To read an object in **Loans 2** under the policy of Figure 3, a user needs to be assigned to Loan Officer or Auditor. We will refer to the group of users that meet this criterion as gr3. To write to an object in **Loans 2**, a user needs to be assigned to Loan Officer. We refer to that group of users as gr4. Now, let us consider the groups that can list and create/delete the children of **Loans 2**.

In general, a user needs to have permissions to list children for all directories along the path to a file for which they have read access. In the case of a representation of a directory of any type, this group would correspond to the users with read access to an object contained in the representation. In the case of **Loans 2,** that is gr3.

Now, let us consider the group of users that can create/delete children. This group of users would correspond to the users that can create/delete objects in **Loans 2**. In accordance with the policy, these users would be required to be assigned to both Loan Officer and Branch 2, namely u3. Given that read on a directory implies list and write on a directory implies create/delete children, we can derive the follow ACL for **Loans 2**.

Loans 2: Custom
file (inherit) – gr3, r; gr4, w
directory – gr3, r (list); gr5, w (create/delete children)
-- where gr3=u2, u3, u4; gr4=u2, u3; and gr5=u3

Because file level permissions apply to children (files) of the directory, ACL file inheritance is specified. Again, due to its designation as "Custom," this ACL file inheritance is blocked for **loan-1,** enabling the preservation of u2's denial to write to loan-1. Using the same approach used for **Loans 2**, an ACL can be created for **Loans 1** and **Accounts 1** that also contain files:

Loans 1: Custom
file (inherit) – gr6, r; gr7, w
directory – gr6, r (list); gr8, w (create/delete children)
-- where gr6=u2, u3, u4; gr7=u2, u3; gr8=u2

Accounts 1: Custom
file (inherit) – gr9, r; gr10, w
directory – gr9, r (list); gr11, w (create/delete children)
-- where gr9=u1, u4; gr10=u1; gr11=u1

Now, let us consider representations of directory repositories that do not contain files. For these representations, a read (list) ACL is required. Given a user needs to have permissions to list children for all directories along the path to a file for which they have read access, the Policy Analytic Engine could simply identify the users who can read an object contained in the representation. Applying this approach to **Loans**, **Accounts**, and **Products**, we formulate their ACLs:

Loans: Custom
directory – gr12, r (list) --
where gr12=u2, u3, u4

Accounts: Custom
directory – gr13, r (list) --
where gr13=u1, u4

Products: Custom
directory – gr14, r (list) --
where gr14=u1, u2, u3, u4

4.3 Creating Host Access Control Data

The method further creates corresponding group(s) as well as user account(s) and an ACL on the local host repositories using the computed group and the ACL of the corresponding representation. Figure 5 depicts the creation of such access control information on a local host system regarding **loan-1**.

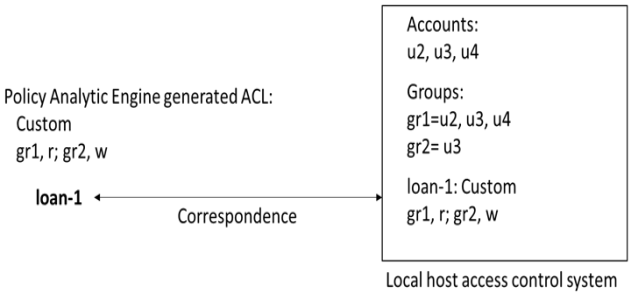

Figure 5: Creation of accounts, groups, and ACLs in local host access control system corresponding to loan-1.

Subsequently to creation of access control information pertaining to **loans-1** the method could create access control information pertaining to **Loans 2**, as shown in Figure 6.

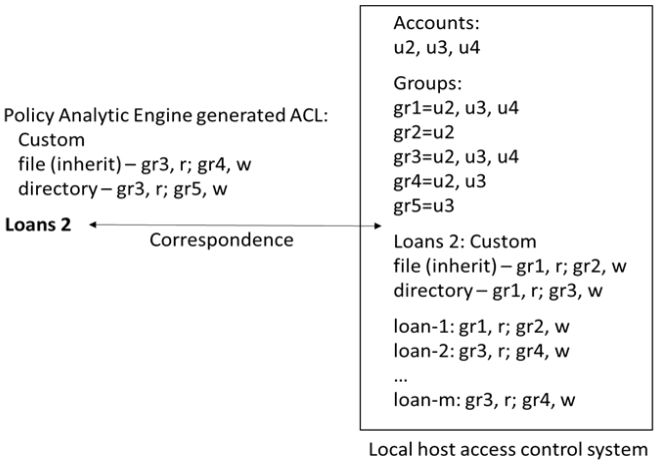

Figure 6: Creation of accounts, groups, and ACLs in local host access control system corresponding to Loans 2.

4.4 Updating Host Access Control Information

As the ABAC policy changes, the method updates appropriate accounts, groups, and ACLs pertaining to affected representations and automatically updates ACLs on corresponding local repositories. Consider the update of the ABAC policy of Figure 3 as indicated in Figure 7.

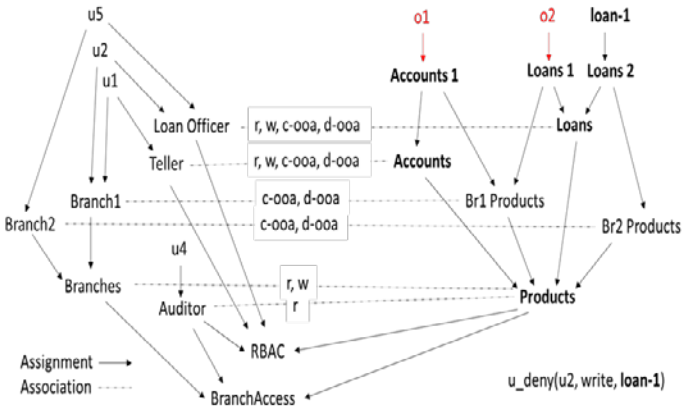

Figure 7: Updating ABAC policy.

Under the updated policy, user u3 has been deleted and replaced by user u5, a new Loan Officer in Branch 2. **Loans 2** is affected by this policy change, and consequently, the logic automatically updates the access control data of the local host access control system as illustrated in Figure 8.

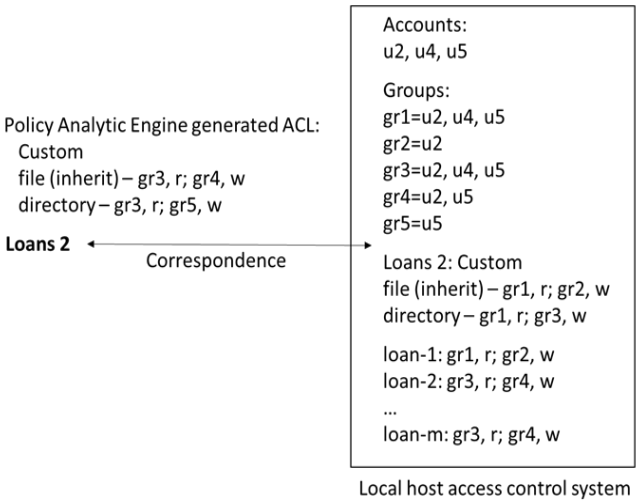

Figure 8: Local changes to the user accounts, groups, and ACLs in correspondence to the updated ABAC policy of Figure 4

5. SYSTEM OPERATION

Operationally, administrators express ABAC policies, introduce representations of local repositories into the policy expression, and instruct the creation of ACLs for repositories through the administrative API of the ABAC Control Center.

Host users attempt to access repositories in local host systems as they normally would, and ABAC policies are enforced in those systems in terms of host ACLs managed by the method. Although the examples used to describe the method pertain to a single local host, the method allows for the centralized management of ACLs in multiple hosts, each within an independent administrative domain as shown in Figure 9.

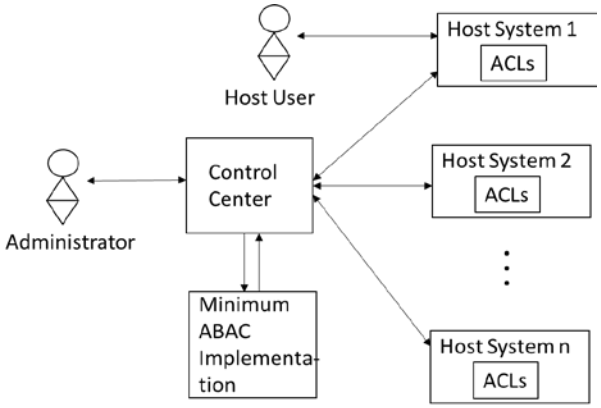

Figure 98: Centralized ABAC policy management and local decision-making and enforcement across multiple security domains

Because of this use of ACLs, access decisions are computed and policy is enforced with an efficiency far superior to an ABAC system that includes PEP and PDP components.

6. RELATED WORK

The method described in this paper is not the only system used for the centralized management of ACLs. In fact, an entire class of products exist, referred to as Enterprise Security Management Systems (ESMSs), which are used for centralized management of authorizations for resources resident in host systems and distributed throughout the enterprise. A common abstraction used by these systems is that of roles and RBAC in general [9, 10]. For instance, roles stored and managed in a directory are used to formulate groups used on ACLs or create ACLs in accordance with role membership and permissions directly associated with roles. The Role Control Center (RCC) [11] is a robust implementation that makes use of much of the entire RBAC abstraction. RCC supports an ESMS model with general role hierarchies, static separation of duty constraints, and an advanced permission review facility (as defined in NIST's proposed RBAC standard [12]). The RCC server is responsible for mapping selected subgraphs of the role graph (called views) to user accounts and groups on heterogeneous hosts as well as for mapping abstract objects and role permissions to actual objects and permission structures (e.g., ACLs) on those hosts. For these tasks, RCC, like our method, uses agent software running on each host to create/delete groups and user accounts, populate the groups with user accounts, and set up ACLs according to commands received from the RCC server. Consequently, RBAC policies are enforced using host ACL mechanisms.

Although there are architectural similarities with RCC and other ESMS products, the method described in this paper is the first to achieve enforcement of ABAC policies using host ACL mechanisms. The enforced policies are based on combinations of user attributes (including but not limited to roles) and object attributes. The ACLs that enforce policy are arrived at not through one-to-one mapping of roles to groups or role permissions to ACLs, but through policy analytics. In particular, the method is based on the determination of a group of users that can access an object or an object in an object attribute with an access right (e.g., read or write) where the source of the group may pertain to a multitude of user attributes.

7. CONCLUSION AND FUTURE WORK

The method described in this paper enables centralized management of ABAC policies for resources repositories distributed throughout an enterprise using host ACLs. It includes a centralized Control Center surrounded by an Administrator, a Minimum ABAC implementation, and Local Host Systems. Administrators express ABAC policies, introduce representations of local repositories into the policy expression, and instruct the creation of ACLs for repositories as administrative commands using the administrative API of the Control Center. The Minimum ABAC Implementation consists of a Policy and Attribute Administrative Point, Policy Analytics Engine, and database for storing ABAC Policies and Attributes. The Control Center maps the authorization data to the various host system ACL mechanisms using the Policy Analytics Engine, through agent software implemented on the host systems. The Control Center, through the Policy and Attribute Administrative Point, may update ABAC policies and/or attributes stored in the database. In such cases, the Control Center instructs the Policy Analytics Engine to re-compute ACLs for affected representations. Using Agent software, ACLs are updated for corresponding repositories in their host access control systems. Operationally, users attempt to access resources in local host systems, and the ABAC policy is enforced in those systems in terms of their ACLs.

All components of the Minimum ABAC implementation are

available as open source. As such, given ABAC policy decision and enforcement are conducted using native host ACL mechanisms, the only components necessary for implementation of the method are the Control Center and host-agent software.

To date we have conducted a variety of experiments to demonstrate the viability of the method, to include development of agent software for the Windows operating system. We plan on development of agent software for other operating environments along with the development of a Control Center component. In addition, we are using a subset of the components available by Harmonia 1.6 to meet the requirements of the Minimum ABAC implementation.

8. REFERENCES

[1] 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*, National Institute of Standards and Technology (NIST) Special Publication (SP) 800-162, January 2014. http://dx.doi.org/10.6028/NIST.SP.800-162

[2] D. F. Ferraiolo, R. Chandramouli, V. Hu, and R. Kuhn, National Institute of Standards and Technology DRAFT (NIST) SP-800-178, A Comparison of Attribute Based Access Control (ABAC) Standards for Data Services, October 2016. http://dx.doi.org/10.6028/NIST.SP.800-178

[3] The eXtensible Access Control Markup Language (XACML), Version 3.0, OASIS Standard, January 22, 2013. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf

[4] NICITS 499 – Information technology – Next Generation Access Control - Functional Architecture (NGAC-FA), INCITS 499-2013, American National Standard for Information Technology, American National Standards Institute, March 2013.

[5] INCITS 526 – Information technology – Next Generation Access Control – Generic Operations and Data Structures, INCITS 526-2016, American National Standard for Information Technology, American National Standards Institute, 2016.

[6] NICITS 525 – Information technology – Next Generation Access Control - Implementation Requirements, Protocols and API Definitions (NGAC-IRPADS), in initial public review (December 1, 2017 to January 30, 2018).

[7] NIST Policy Machine Versions 1.5 and 1.6 - Harmonia [Website].

[8] Peter Mell, James Shook, Richard Harang and Serban Gavrila, *Linear Time Algorithms to Restrict Insider Access using Multi-Policy Access Control Systems*, Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications, vol. 8, num. 1, March 2017, pp. 4-25, URL: http://isyou.info/jowua/papers/jowua-v8n1-1.pdf.

[9] Enterprise Security Architecture using IBM Tivoli Security Solutions (2002) – IBM Corporation

[10] Enterprise Security Station – User Guide (Windows GUI) – BMC Software Inc., 2002.

[11] Ferraiolo D, Chandramouli R, Ahn GJ, Gavrila SI (2003) The role control center: features and case studies. Proc. of the 8th ACM symposium on access control models and technologies, Como, Italy, pp12–20, June.

[12] D. Ferraiolo, R. Sandhu, S. Gavrila, R. Kuhn, R. Chandramouli, Proposed NIST standard for role-based access control, ACM Transactions on Information and Systems Security 4 (3) (2001).