POLICY-BASED SECURITY MANAGEMENT FOR ENTERPRISE SYSTEMS

R. Mukkamala, L. Chekuri, M. Moharrum, and S. Palley

Abstract With the increasing growth in global enterprises and collaborations among the enterprises, security and trust have become essential for information systems. For example, within an enterprise, there may be a need to maintain security within each project group so the information sharing among the groups is controlled. Similarly, there may be a need to facilitate controlled and timed sharing of data among cooperating enterprises (e.g., coalitions). In this paper, we propose a policy-based security mechanism for such sharing in an enterprise. In particular, in our system, each user (or administrator) specifies restrictions on the use of resources at a particular node (or machine) in terms of a set of policy statements (NRPS and NTPS). Similarly, the owner of each object specifies the conditions on which certain operations can be performed on the object (ORPS and OTPS). Trusted policy enforcement agents (PEA), running at each node in the enterprise (or coalition), ensure that both node and object policies are enforced in the system. We show how the proposed system facilitates dynamic control at object-level and machine-level.

1. INTRODUCTION

With reduced funding for research and infrastructural support, and with increase in cost and complexity of research projects and product developments, more and more organizations are teaming up with other organizations to accomplish their goals. Such collaborations are typically limited in scope as well as duration. For example, military coalitions among several nations are formed to achieve a specific task, and are dissolved at the completion of the task. Similarly, there is an increasing focus on forming grids where enterprises are attempting to share their resources such as supercomputers, compute clusters, storage systems, data sources, instruments, and people to accomplish complex tasks with limited resources

at each enterprise [9,14]. In addition to these formal groups, informal peer-to-peer (P2P) groups are also being formed to share resources [10]. Napster [10], Gnutella [11], and Groove [13] are example P2P groups formed to share resources such as music and software.

While the technologies such as CORBA are helping to enable easy access to resources in distributed and heterogeneous environments [8], managing security and trust remain to be challenging issues [4]. For example, Galiasso et al present policy machine and policy meditation architecture for coordinating diverse policies (e.g., DAC, MAC, RBAC, SoD, work-flow, Chinese Wall and N-person control) in large information networks [4]. Similarly, Hu et al have proposed an open system architecture, the policy machine (PM), to coordinate different policies [6]. This PM relies on the separation of access control mechanisms from access control mechanisms [6]. A formal model to accomplish the goal of automated trust negotiation is proposed by Yu et al [15]. Biskup and Karabulut proposed a hybrid model for PKI to accomplish the task of specification and enforcement of permissions in distributed systems [1]. Work in policy specification languages and formalisms are well researched [e.g., 5,7,12].

In this paper, we propose a policy-based security mechanism for sharing resources within an organization (e.g., across projects or departments) and across organizations (e.g., in a grid or in a consortium). In particular, each user specifies restrictions on the use of resources at his/her node in terms of a set .of policy statements. Similarly, the owner of each object specifies the conditions on which certain operations can be performed on the object (including its copy). Trusted policy enforcement agents, running at each node, ensure that both node and object policies are enforced in the system.

The paper is organized as follows. Section 2 describes our model for the coalition system. Section 3 describes the different node and object policy specifications in the systems. Section 4 provides details on the proposed policy enforcement agent (PEA). Section 5 provides some key implementation details of our prototype of the proposed architecture. It deals with both policy representations and PEA implementation. Finally, section 6 summarizes the paper and discusses future work.

2. SECURITY MODEL

In this paper, for simplicity, we use the terms node and user synonymously. All participating nodes run trusted policy enforcement software (described in section 4). They are connected to other nodes through a network (e.g., Internet, intranet, Milnet). Our model uses the following concepts.

Node and object policies. *Node policies* express constraints on what can be stored or executed at a node (details in Section 3). Similarly, *object policies* express restrictions about a node that can store/access a specified object and the type of operations that may be executed on it (details in Section 3).

Node policy database. Each node maintains a *node policy database*[1]. It contains its own nodal policy as well as the node policies of node that registered with it (or those that it is allowed to communicate with)

Object policy database. The policies associated with objects that a node contains (either owned or cached) are stored in the object policy database at that node. It contains object policy statements for each object that the node holds. Each node, before caching or permanently storing an object, checks with the object policy and the node policy to make sure that they permit the action. Similarly, each request for access (by local user) or for transfer by other nodes also involves checking the object policies and node policies (if request is from other nodes).

Registered node database. In order for a node to send a request (for an object) to another node, both nodes should have mutually authenticated each other. We refer to this process as registration. By mandating node registration prior to object exchange, we enhance the trust among the nodes. Each node stores the set of its registered nodes and their authentication information in the registered node database.

Encrypted storage. All objects are stored in an encrypted fashion. Each object has a separate encryption key maintained by the system.

Encrypted sessions. All communication between mutually trusted nodes take place via encrypted sessions. A one-time key is used for each such session.

Policy-enforcement software. Security enforcement is done using trusted and certified software. We refer to it *as policy enforcement agent* or PEA. All interactions with a node, internal as well as external, are through this software only. (Details are discussed in section 4.)

Anonymity. To maintain requester anonymity, request to a node do not carry any information about the originator of the request (i.e., the root of the nested tree). Instead, a node can only know about the last node in the request chain. Further, a node does not reveal the identity of its requester when forwarding it to the next node (its trusted neighbor). Finally, when the object is downloaded from a leaf (of the nested tree), the object takes the same as the request path, to maintain anonymity.

Object caching. In order to improve efficient sharing, a node on a request path is permitted to cache an object if its nodal policy and the object's policy permit it (details in section 3). In other words, in the interest of other nodes,

[1] Here, the term *database* is used in a generic sense to represent a collection of information not necessarily a relational or object database.

each node reserves a portion of its object storage to cache objects that transit through it. The cached objects are referred to as *transit objects* and those stored permanently are referred to as *resident objects.*

3. POLICY SPECIFICATION

In this section, we first describe the classification labels assigned to nodes and objects based on their requirements or characteristics. We then describe policy statements that use the labels to express the restrictions of access (e.g., read, write, execute) and transfer placed by objects and nodes in the system.

3.1 Classification labels

We specify the characteristics of nodes and objects that register with the nodes in terms of a set of labels. The characteristics include the entity preferences including security, reliability, and performance. A node's owner (user or administrator) specifies node labels at the time the node joins the coalition system. Similarly, the owner of an object specifies object labels at the time the object is uploaded into the coalition system. While a node's labels indicate the types of services it offers (e.g., confidential class, high-reliability, giga-byte storage, scientific processing, high-bandwidth, etc.), an object's labels indicate what it requires for storage, execution, or transfer (e.g., secret class, 128-bit AES encryption, large memory, FFT-library, etc.). Following is a brief description of the labels.

Security label (SL) for a node specifies the security classification of a node in terms of terminology accepted in the coalition system that it has joined. For example, if the system uses role-based access control, then they may use the respective roles such as president, vice-president, manager, clerk, etc. Similarly, in a military coalition with MLS-based system, nodes may be labeled as unclassified, confidential, secret, and top-secret. In coalitions where different organizations use different classification, we need a trust negotiation or meditation systems (e.g., [4,15]). Similarly, each object in the coalition system is also assigned a security label based on its criticality and intended use. In general, the security labels are used to restrict access to the nodes and objects. The restrictions are expressed through policy statements as described in section 3.2.

Reliability labels (RL) are similar to security labels. A node's reliability label indicates its degree of reliability (or availability). For example, a node that is always on-line and available may be labeled as a high-reliability node. Similarly, a node that is connected intermittently may be labeled as low-

reliability node. A mobile node (say an executive's laptop) with limited power-supply may be indicated as a very-low-reliability node. As before, we assume that a coalition system has a standard set of reliability labels and their definitions. Otherwise, as above, a negotiation mediation system needs to be used. Optionally, each object may also be assigned a reliability label depending on its requirements.

Performance labels (PL) are similar to reliability labels. For example, the performance label for a node indicates the type of performance it offers. For example, a node may be labeled as heavy-load if it is highly utilized. Similarly, it may be assigned large-disk-space and medium-processor labels depending on its resources. An object is also assigned performance labels depending on its performance requirements. A large database object may be labeled as large-size, DB-object.

Depending on the coalition system and its domain of applications (e.g., scientific, military, governmental), several other labels may be assigned to nodes and objects.

3.2 Policy Statements

Policies describe the limitations (or constraints) on the use/handling of the entities (objects and nodes) under different conditions and at different times, and hence help us in enforcing security in a coalition system. In this paper, we consider object policies and node policies.

Let us first consider object policies. In a coalition system, once a user creates an object and uploads it (via trusted policy enforcement software), other nodes could send requests for it. In such a case, an instance of the object may transit through several nodes (of the coalition system) prior to reaching the end-node that made the. In such an end-to-end transfer, we refer to the two end nodes as the *source node* and the *destination node,* respectively. They are also referred to as *terminal* nodes. The remaining nodes involved in the transfer are referred to as *transit* nodes. Thus, each object instance may be at a terminal node or at a transit node. The limitations placed on an object while it is at a terminal node or at a transit node are expressed in terms of two policy statements, *Object Residential Policy Statement* (ORPS) and *Object Transit Policy Statement* (OTPS), respectively.

- The *Object Residential Policy Statement (ORPS)* specifies the constraints placed on an object usage while residing at a terminal node (as a permanent object, and not as a cached object). In addition, it specifies the requirements that a node has to satisfy to maintain a copy of the object (uploaded at user's request). For example, an object may require that it be executed at a high-performance node or a node with secret classification or

above. The object usage constraints specify the rights that the local node has on the object. Figure 1a illustrates a simple ORPS policy for an object. It states that a node at a security level of confidential or higher may only read the object. But a node at a level of secret or higher with a high-performance node can both read and execute it.

- The *Object Transit Policy Statement (OTPS)* specifies constraints placed on an object at a transit node (when an object is cached during transit, it is considered as a transit object). For example, it may specify the attributes (e.g., node classification level) that a node should possess for the object to transit to it. It may specify rules on transfers such as using a specific type of encryption prior to transmission to the next node along the path, etc. It may also specify the encryption standard to be used while being stored in the local transit cache. OTPS may specify policy rules about delegation and exportable rights. Figure 1b illustrates a simple OTPS policy. It states that only nodes with top-secret classification, nodes with secret classification and reliability label of medium or higher, and nodes at confidential classification with high or better reliability can act as transit nodes for the specified object. Thus, the object is restricting itself from going through any arbitrary node.

While the owner of an object sets the above two policies, the owner of a node (user) sets similar policies referred to as NRPS and NTPS, describing a node's behavior as a terminal node (for resident objects) and as a transit node (for transit objects), respectively.

- The *Node Residential Policy Statement (NRPS)* specifies conditions that an object has to satisfy to be stored as a resident object (and not as a cached object) at a node. These are the objects that are uploaded at user's request and, hence, accessible to the node(user). The policy specifies the types of objects that are permitted to reside at a node either directly uploaded by a local user or transferred from another node (at user's request). In implementing NRPS, the system will act as a firewall restricting the type of objects that can reside at a node permanently. Figure 1c shows a simple NRPS policy. Here, the node specifies that only three types of objects can reside at it: (i) objects with classification of confidential or lower, with a medium or lower reliability requirement, and size of 50k bytes are less; (ii) objects at secret or above with medium or lower reliability requirement; (iii) objects at secret or above classification with reliability requirement of high or above, and less than or equal 10k bytes in size. Objects that do not satisfy these requirements may not be stored at this node.
- *Node Transit Policy Statement (NTPS)* specifies constraints on the type of objects that can transit through a node. In addition, it may also restrict the type of source nodes from which objects may be transferred or the type of encryption that should be used for an object to pass through the node. The policy is useful in restricting other users from abusing certain

nodes as the path for their transfers. As transit nodes usually keep a copy of the transiting objects, restricting such traffic has relevance in node security and performance. Considering the example in Figure 1d, the node transit policy specifies that only objects with secret or higher security classification and a reliability requirement of medium or lower can be received and be forwarded.

In addition, in order to be able to implement some policies such as the Chinese wall policy and the workflow model, we have introduced the object history (or event-response) policy.

- *Object History Policy Statement (OHPS)* specifies new constraints to be placed (either on this object or other objects) when an action (operation) is performed on an object. This is similar to the historical-related policy in [6]. This policy is also supported by a log or history of all operations on that object at that node.

4. POLICY ENFORCEMENT AGENT (PEA)

In order to enforce the system-wide policy-based security, each node of the coalition system should enforce the policies. We propose that each node run a policy enforcement agent (PEA) on top of its operating system. PEA is trusted, digitally signed, software. A node, prior to its joining a coalition system, must download and install the PEA. It may be available either from the coalition administrator or through a trusted third-party. It represents a gateway through which all coalition traffic to and from a node must pass through. Traffic may be either the exchange of objects and policy statements, or simple requests/replies among the coalition nodes. Even local user's requests for local (shared) data must go through PEA. Thus, the PEA becomes the only gateway we need to protect. Primarily, PEA keeps track of three types of entities to enforce security.

- **User credentials.** A user (at a node) has no direct access to his/her credentials as they are stored securely, in an encrypted form, using a system generated symmetric key. A user can only use them through PEA's client interface.
- **Objects.** All objects (both resident and transit) are kept in an encrypted form in an *object vault.* Objects are encrypted using individual system-generated object keys. Users can only access the objects has through the client interface of the PEA.

SL	PL	RL	Read	Write	Execute
≥Confidential	*	*	Yes	No	No
≥Secret	≥High-Performance	*	Yes	No	Yes

Figure 1a. Example ORPS policy

P	SL	PL	RL
1	Top-Secret	*	*
2	Secret	*	≥ Medium
3	Confidential	*	≥ High

Figure 1b. Example OTPS policy

P	SL	PL	RL	Size
1	≤ Confidential	*	≤ Medium	≤ 50k
2	≥Secret	*	≤ Medium	*
3	≥Secret	*	≥ High	≤ 10k

Figure 1c. Example NRPS policy

SL	PL	RL	Receive	Send
≥Secret	*	≤ Medium	Yes	Yes

Figure 1d. Example NTPS policy

- **Policy Statements.** Node and object policy statements are stored separately in a *node policy database* and an *object policy database,* respectively, in an encrypted form, using a system-generated key. The object history policy statements are also stored in the object policy database. Users have no direct access to the policy database. In addition, the following procedures are observed.
  - Any object and policy statement transfers take place via a secure session using a disposable (one-time) session key.
  - When a node receives an object, it also receives both ORPS and OTPS of that object. The object is then encrypted (with a system-generated object key) and stored in the object vault. All policy statements are stored in the encrypted policy database.

We now discuss the protocol followed by the description of the proposed architecture.

4.1 The PEA protocol

A PEA at a node interacts with its user as well as with PEAs at other nodes. Following is a brief description of our PEA protocol.

1. When introducing a new object, a node (user) is considered as the owner of that object. Along with the object, it should introduce the two object policy statements (ORPS and OTPS). The PEA of the owner node digitally signs the policy statements (with its private key). The (encrypted) objects are stored in the object vault and (encrypted) policies are stored in the policy database.
2. When a node's PEA contacts another node's PEA for the first time, a PEA-PEA authentication takes place. If the authentication succeeds, each node adds the other to its *node registration set* and they exchange their NTPS. A node can send/receive a request to/from another node only if it is in its node registration set. (Details on how nodes discover each other is outside the scope of this paper.)
3. When a PEA receives a request (from its local user) for access to an object, there are three cases to be considered. (i) If the object is in its object vault as a resident object, then it checks its ORPS to allow or deny access. (ii) If the object is in its object vault as a transient, it checks with its own NRPS to check whether or not it can be made resident. If NRPS permits it, then the node is made resident and then the node's ORPS is checked to see if the requested operation is permitted. (iii) If the object is not in the object vault, then a request for the object is sent to one (or more) of its registrant nodes.
4. When a PEA receives a request from another PEA for an object, there are two cases to consider. (i) The requested object is either a resident

or a transit object. Hence, object's OTPS and the requesting node's NTPS are checked to see if the transfer is permitted. If permitted, the object (along with its policies) is sent (in an encrypted form) to the requestor. If not permitted, a reply of denial is sent. (ii) The requested object is not in the object vault. So the request is forwarded to one (or more) of its registered nodes.

5. Transit objects are considered as cached objects. Hence, they are removed from the object vault based on the object caching policy and the storage limitations. Resident objects may be removed at user's request.

The PEA architecture will keep the system running securely as long as each PEA at each node follows the above protocol.

4.2 The PEA Architecture

Figure 2 depicts the proposed PEA architecture for secure coalition systems. The architecture is described in terms of an individual PEA and its components.

User API (omitted in Figure 2). This interface enables users to perform the following tasks.

- *Authenticate the user (e.g., password, smart card, etc.).* While this may be customized for individual PEAs, a node's policy should clearly state the methods that it accepts for authentication. This information is used when PEA's authenticate each other during registering with each other. For example, a node may only accept to register nodes that use smart cards for user authentication.
- *Request access on objects.* If an object is not at this node, PEA initiates the process to bring it from other PEAs.
- *Add/delete objects.* User may request to upload new objects to be shared or remove existing objects.
- *Add/modify/delete policy statements* for resident (but not transit) objects.

Authenticator. Each PEA keeps a database of all nodes that have registered with it. It can accept requests only from such nodes. The authenticator performs the following tasks.

- *Authenticate a new node.* Upon receiving a registration request from an unknown node, the recipient's authenticator verifies the credentials that the new node presents. Upon authentication, the registration request is either accepted or denied.

- *Authenticate the sender of a request.* Whenever a PEA receives a request (e.g., for an object that it has), the authenticator first verifies that it is in its registered node set and verifies its credentials.
- *Maintain malicious node list.* The authenticator is also responsible for maintaining a list of *malicious nodes* that either it has noticed or other PEAS have informed it.

Policy databases. The three policy databases (object policy, node policy, and meta policy) are stored in an encrypted manner by the PEA. It decrypts the policies on retrieval and encrypts them while storage. As explained in the next section, there is a single policy file for each object and for each node. The history-based policies are stored along with the object policies.

Policy checker. This module checks if an incoming request from a node (local or remote) is consistent with both the node and object policies. If it finds that there is a conflict, it either rejects the request (when it conflicts with both) or sends it to conflict resolver (when one of them agrees and the other denies). All requests that are found to be consistent are forwarded to history policy updater.

Conflict Resolver. This module resolves any conflicts between node policies and object policies in the process of request processing. It makes use of the meta policy database, either set by the coalition or by the agreement between individual users.

History-policy updater (omitted in Figure 2). This determines the responses (if any) in case an accepted operation were to be performed on an object. If it determines that there is a conflict with the new policy to be added and the existing node policy, it refers it back to the conflict resolver. If accepted, it adds the new policy to the object database.

Object vault interface (omitted in Figure 2). The object vault (OV) is an encrypted database that stores all objects that a node receives, either as resident or as transit. The OV interface enables a PEA to interact its *object vault.* The interactions involve the following tasks.

- *Encrypt and store new objects.* Using appropriate key, encrypt newly introduced objects before storing into the Object Vault.
- *Decrypt and retrieve existing objects.* Using appropriate key, decrypt objects after retrieval from the object vault.
- *Delete objects.* Delete expired or rarely retrieved objects from the object vault according to its predefined policy. An object may be deleted at user's request also.

Key store (omitted in Figure 2). The key store is an encrypted database that keeps track of all keys that a PEA uses. The contents of the key store are themselves encrypted with a strong symmetric key derived from some user-supplied authentication information (e.g., password). It stores object encryption keys, policy database keys, and registration database key.

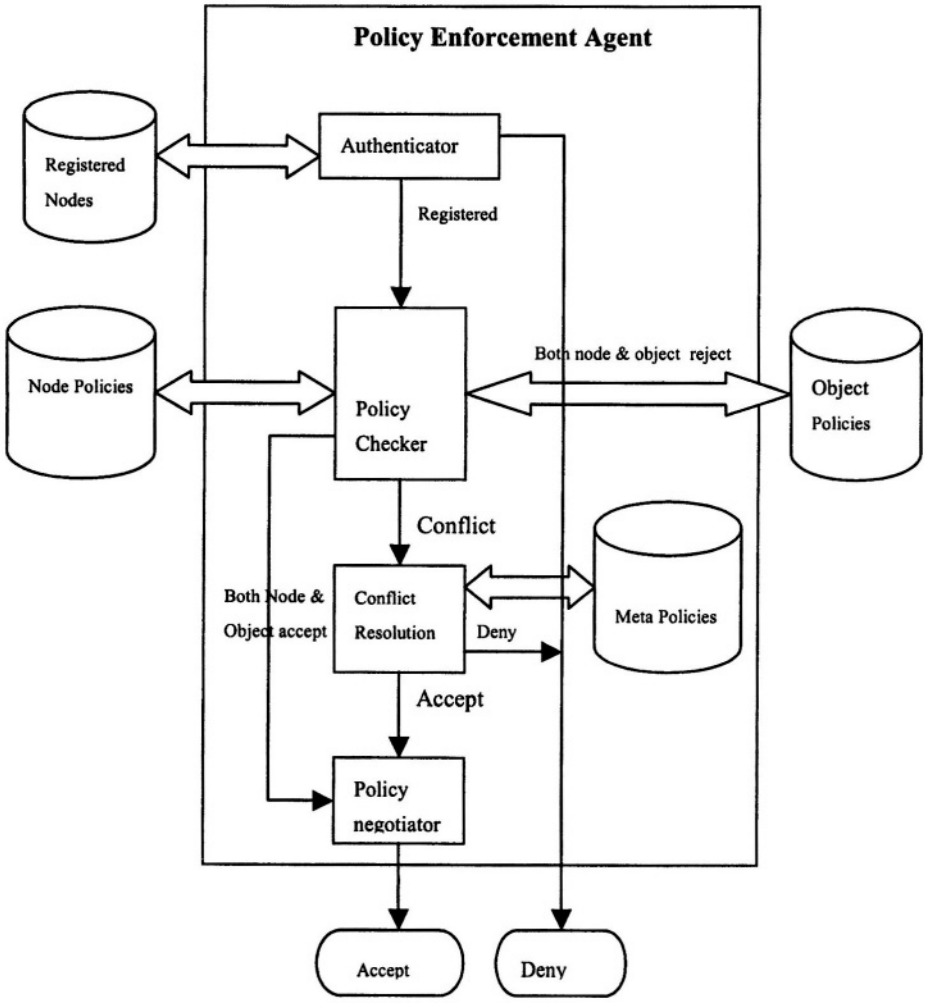

Figure 2. PEA Architecture

5. IMPLEMENTATION

We have implemented a prototype system of the proposed coalition security architecture. The system was tested with several nodes (within the CS department) in the coalition and tens of objects. Following are some of the implementation details.

User (node) authorization with the coalition PEA. In the current system, we use a login and password for PEA authorization of a node. This could be enhanced by systems such as one-time passwords, smart cards, etc. Adding an object: A user can browse through the objects to be uploaded to the coalition system. For each of these objects, security, reliability and performance labels need to be assigned. The permissions for read, write, and execute as well as the expected levels of security, reliability and performance levels of the nodes may be specified. For more flexibility, we allow qualifiers such as <, >, <=, >=. Similarly, the user may create the object transit policy. The PEA software converts the policies into XACML and stores in the object policy database.

Adding History-based policies. A user can also specify the event-responses associated with an object at the time of uploading an object.

XACML Representation of the policies. Due to the easy portability and the library support provided by XACML (e.g., PEP and PDP), we have represented all policies in XACML. The PEA software generates the XACML file automatically when the user enters data through the GUI.

Authenticator. The authenticator checks if the requesting nodes have registered or not. If so, it may further conduct other security authentication checks (e.g., challenge-response, login-password, etc.)

Registered Nodes. Each node maintains a database (a list) of registered nodes. It has the authority to register other nodes from where it can receive objects and requests for objects. Whenever a node is registered, a meta-policy describing the relationship between these two nodes is also created.

Policy checker. The main purpose of the policy checker is to compare the request and the policies and to decide whether or not to allow the request to execute. First, the received request (in native format) is converted to an XACML file. The policy retriever retrieves all related node policies (in XACML) from the (node) policy database. The entity called Policy Enforcement Point (PEP) forms a request (using the XACML request language) based on the attributes of the subject, action, resource, and other relevant information. The PEP then sends this request to a Policy Decision Point (PDP), which examines the request and the policies (written in the XACML policy language) that are applicable to this request, and determines whether or not access should be granted. Similar checks are made with the request and the object policies. The final decision is expressed in XACML format. If both node and object policies permit it, the decision is sent directly to policy negotiation component of the PEA. In case of a conflict, the decision is sent to the conflict resolution module.

Conflict resolution. In case of a conflict between the node policy and the object policy with respect to a request, conflict resolution module uses a meta policy database (which may have been agreed upon either by the coalition or by individual nodes at registration) to resolve the conflicts.

History-policy. Once the policy checker/conflict resolution modules return 'permit' as a result, the PEA checks Object's historic-policy database to check for any possible matching actions to be performed. If there's a match, it checks if the generated response and its corresponding modification to the object policy database would conflict with the node's policy. In case of such a conflict, the PEA has to once again go back to the conflict resolution module.

6. CONCLUSION AND FUTURE WORK

In this paper, we introduced a policy-based security for supporting secure-coalitions among enterprise systems. We defined the constraints placed on object access and node usage in terms of policy statements. Policy enforcement agent (PEA), which is trusted security software that runs on peer node's operating system, enforces the policies. We suggested an architecture and showed some of the details of our prototype implementation. We used XACML as the policy specification language. Finally, the ability of the proposed system to thwart insider and outsider attacks were discussed through several scenarios. The main issue that we need to address in future work is performance. Especially, we need to compare the performance of the proposed system with respect to other current systems. We propose to carry out such a performance study. In addition, we propose to test the completeness of the proposed policy scheme by modeling more realistic systems and determine the efficacy of the system.

References

[1] J. Biskup and Y. Karabulut, "A hybrid PKI Model: Application to secure mediation," pp. 271-282, *Research Directions in data and applications security,* Kluwer Academic, 2003.

[2] S.Dawson, S. Qian, and P. Samarati, Secure interoperation of heterogeneous systems: A mediator-based approach. *Proc. 14th IFIP TC-11 International Conference on Information Security,* Chapman and Hall, 1998.

[3] G. Edjlali, A. Acharya, and V. Chaudhary, "History-Based Access Control for Mobile Code," *Proc. ACM Conference on Computer and Communications Security,* pp. 38-48, 1998.

[4] P. Galiasso, O. Bremer, J. Hale, S. Shenoi, D.F. Ferraiolo, and V.C. Hu, "Policy Mediation for Multi-Enterprise Environments," *ACSAC 2000,* pp. 100-106, 2000.

[5] J.A. Hoagland, R. Pandey, and K.N. Levitt, "Security policy specification using a graphical approach," *Technical Report CSE-98-3,* The University of California, Davis, July 1998

[6] V. Hu, D. Frincke, and D. Ferraiolo, "The Policy Machine for Security Policy Management," *Proc. International Conference on Computational Science,* pp. 494-506, 2001.

[7] S. Jajodia, P. Samarati, M. Sapino, V.S. Subrahmanian, "Flexible support for multiple access control policies," *ACM Trans. Database Systems,* Vol. 26, Issue 2, pp.214 – 260, 2001.

[8] T.J. Mowbray and R. Zahavi, *The Essential CORBA: Systems Integration using distributed objects,* John Wiley, New York, 1995.

[9] N. Nagaratnam, et al., *Securirty for open grid services,* GWD-I, OGSA Workgroup, July 2002.

[10] A. Oram (Editor), *Peer-to-peer: Harnessing the benefits of a disruptive technology,* O'Reilly, 2001.

[11] M. Ripeanu, " Peer-to-Peer Architecture Case Study: Gunutella," *Proc. of 2001 Conf. On Peer-to-Peer computing,* Linkoping Sweden, 27-29, August 2001

[12] F.B. Schneider, "Enforceable security policies," *ACM Press,* New York, NY, USA, Volume 3, Issue 2, February 2000.

[13] J. Udell, N. Asthagiri, and W. Tuvell, "Security," in *Peer-to-peer: Harnessing the benefits of a disruptive technology,* A. Oram (Editor), pp. 354-380, O'Reilly, 2001.

[14] Welch, et al., "Security for grid services," Proc. *Twelfth International Symposium on High Performance Distributed Computing (HPDC-12),* IEEE Press, 2003

[15] Yu, T., Winslett, M., and Seamons, K.E., "Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation," *ACM Trans. Information and System Security,* Vol. 6, No. 1, pp. 1-42, Feb. 2003.