Dynamic policy based model for trust based access control in P2P applications

Madhumita Chatterjee, G.Sivakumar, Bernard Menezes
Indian Institute of Technology
Department of Computer Science and Engineering
Mumbai, India 400 076
Email: madhumita@cse.iitb.ac.in, siva@cse.iitb.ac.in, bernard@it.iitb.ac.in

Abstract—Dynamic self-organizing groups like wikipedia, and f/oss have special security requirements not addressed by typical access control mechanisms. An example is the ability to collaboratively modify access control policies based on the evolution of the group and trust and behavior levels. In this paper we propose a new framework for dynamic multi-level access control policies based on trust and reputation. The framework has interesting features wherein the group can switch between policies over time, influenced by the system’s state or environment. Based on the behavior and trust level of peers in the group and the current group composition, it is possible for peers to collaboratively modify policies such as join, update and job allocation. We have modeled the framework using the declarative language Prolog. We also performed some simulations to illustrate the features of our framework.

I. INTRODUCTION

A P2P group can be thought of as a system of processes that evolves and adapts according to interactions and relationships within the group. Self-organizing groups are those that have the tendency to generate new patterns spontaneously. For example a work team will generate norms, structures and procedures based on the behavior of the members of the group, maybe even evolving its original mission. In unpredictable environments groups must adapt to survive. Thus there is a need for self-organizing groups to change the access control policies in the interests of the group. These changes should be within the frame-work of a well defined group charter. Further in such dynamic P2P communities where peers are unknown to each other and uncertain about each other’s reputation it is necessary to develop strategies for establishing trust among peers. Trust [5], [6] is a means for understanding and adapting to the complexity of the environment. A motivating example is the self-organizing P2P group, Free and Open Source Software (F/OSS) [8] which represents an approach for communities of like-minded participants to develop software systems and related artifacts that are intended to be shared freely, rather than offered as closed commercial products. Participants within these communities often participate in different roles like core developer, module owner, code contributor, code repository administrator, reviewer or end-user and communicate information about their content updates via online discussion forums, threaded email messages, and newsgroup postings. F/OSS systems co-evolve with their development communities. A F/OSS project with a small number of developers (most typically one) will not produce and sustain a viable system unless/until the team reaches a larger critical mass of 10-15 core developers.

In this type of a decentralized and dynamic peer group scenario it is necessary to have an adaptive access control mechanism where peers have the right to collaboratively make access control decisions, modify these decisions and frame rules based on attributes as well as trust evolved in the group. We propose a hybrid access control framework which integrates policy based access control and trust based access control. We introduce the concept of multiple levels and unequal roles for peers at each level in self-organizing groups. Peers in one level should be able to dynamically elevate their levels in a group. Further they should have the right to collaboratively modify the access control policies governing their levels and the levels below them. Our prolog based access control framework addresses all these issues and we validate it by experiments.

II. SYSTEM MODEL

We consider a peer based model where every peer has a unique user identity associated with it. A group is a set of peers that are governed under a set of rules that describe minimal conditions to be part of the group and is formed based on a particular interest criterion. These initial or minimal set of rules or access policies form the group charter. For a group to survive it must adapt to the environment and to the behavior of the peers. Thus it is necessary for peers in a group to be able to dynamically modify the access control policies based on evolving trust parameters. The changes must however be within the groups’ constitution charter, so that loosening of access control is not permitted to such an extent that the group performance degrades. Further malicious peers should not be allowed to change the group’s policies.
The group has an admission control policy [1], [2] based on a dynamic threshold. The admission policy is enforced using voting. A group membership certificate is issued to a joining peer which contains tokens signed by the existing peers. A new peer wishing to join a group, has a maximum intrinsic potential and declares an initial self proclaimed rating which is considered to be his initial trust level. Roles are assigned to peers based on this initial rating. Once in the group peers are periodically rated by other peers in the group and signed rating certificates are saved by some of the peers having higher functionality. Trust value of a peer at any point of time is calculated based on the recommendations that he has received from other peers as well as the trust value of the recommending peer. Each time a new peer joins or an existing peer leaves, group re-keying is done so as to ensure backward and forward secrecy. The framework defines policies for a new peer to join, or leave and for an existing peer to update its level. The access policy can decide the role a peer assumes, based on the request of the peer, and his initial rating and credentials. A peer can be updated depending on his behavior and rating calculated by the other peers.

A. Basic Assumptions

The following are the assumptions made in our model.

- No centralized control: Traditional access control models such as ACL or RBAC [3] rely on central servers for authorization. In our model a peer has a high level of autonomy and can frame and manage his own policies.
- Peers in a group can belong to different levels within the group and can perform several roles. A role of a peer is independent of the level which the peer is in.
- Peers are individually capable of performing tasks of authentication, voting, access control, key management etc.
- Peers can compute their own public-private key pair and provide self signed certificates, binding their identity with their public key.

Each group has a group policy, which describes the roles permitted in the group and the rules associated with each role. A well formed group charter could have some initial rules as follows:

- Well defined rules for adding new peers and discarding malicious peers.
- Members should be able to share resources that exist among different peers based on the trust value associated with the peers.
- Framing group policies like minimum trust levels to be maintained.
- Rules for upgrading trust level of a peer or degrading trust level.

- Rules for creating and destroying a group.

The functions that a peer is capable of performing in a group are storing and verifying certificates, authentication, voting, updating of levels, key management etc. The extent to which a peer performs these tasks is decided by the role the peer wants to play in the group. We identify the following roles:

- Member peer A peer in this role is a minimal functionality peer who participates in the normal group activities but does not contribute to the admission of new peers, nor updating of levels of existing peers.
- Admission Control Peer A peer with this role would participate in the voting algorithm, and would therefore store public keys of every other peer. However it does not need to store the updated ratings.
- Maximal role peer A peer in this role would have all the functional components and would participate in updating levels of existing peers also.

III. HYBRID ACCESS CONTROL MODEL

Our hybrid access control model is based on an integration of policy based and trust based access control [3], [4], [7]. The trust parameters in our model build upon the PeerTrust [10] model. We specify context-based attributes for specifying trust and assign specific weights to these attributes. We further define a tunable trust metric to combine all trust parameters viz: direct trust, indirect trust based on recommendations and credibility of the peer. As peers perform in the group, trust of each peer evolves based on the interactions. A role represents a set of privileges and rights. It involves meta-level policies in regard to admission of users to the role. Integration of the roles, tasks, privileges and security policies needs a proper framework for policy specification. We are able to express dynamic rules at runtime, as well as change existing framework depending on the status of the group at runtime through Prolog.

A. Policy Driven Trust Based Access Control (PDTBAC)

Our model focuses on the genetic evolution of groups based on group composition. In PDTBAC it is possible to deploy access control policies flexibly and dynamically. Based on the behavior and trust level of peers in the group and the current group composition, it is possible for peers to collaboratively modify policies governing their level. The group policy can also dynamically prioritize requests for join. Join priorities would depend on current group composition. Thus low level requests for join could be postponed in a group which already has a large number of members performing the lower roles. Peers in one level are also allowed to be dynamically updated to a
higher level. Updating policy also depends on current group composition and behavior. A peer could also be collaboratively ejected from a higher level to a lower level or even out of the group if his behavior in the group degrades. It is also possible for peers in the highest level to introduce a new level in the group if the situation so demands. Some policies can have higher priorities than others. Policies can apply to different levels of the system and can be global or apply to only parts of the system. To the best of our knowledge a group with such enhanced peer capabilities has not been discussed so far in the group communication systems.

B. Modeling Policies and their dynamic environment

In policy driven TBAC, the policy description language besides being expressive and easy to use must be flexible enough to allow extension of a policy by adding or modifying its constraints easily, without affecting existing policies. We use a logic programming system to realize our access control policies. We capture the policies using Prolog rules. A prolog rule is an expression of the form

\[ Ro(uo) :- R1(u1), \ldots, Rn(un) \]

where Ri are predicates and ui are (possibly empty) tuples of variables and constants. The head of the rule is Ro(uo) and the sequence of formulae on the R.H.S is the body of the rule. We chose Prolog as our policy description language because it is declarative, supports back tracking and can express non-deterministic constraints. It is possible to reason from a set of Prolog rules, it also supports meta-level reasoning, thus making policy conflict detection possible. Dynamic rules can be modeled using the assert and retract clauses. New facts from independent policy sources can be added to the policy base before decisions are made, ensuring dynamic decisions at runtime. A policy interacts with its dynamic environment by consulting facts in the environment and constraining certain actions in the environment. In order to ensure that totally new policies outside the original group charter cannot be framed dynamically, we maintain a hash value of the original group charter with every peer. Thus a peer can at any time verify whether the policies being applied are as per the constitution framework.

The system is modeled using state transition diagrams. The state model is shown in figure 1. At a specific instant the system is in state \( S_i \) where policy \( P_i \) applies and group events like Join, Leave, Update, Review, Submit, Moderate and so on keep occurring. These group events have been labeled in the figure as E1, E2, E3. The inputs to the system are Control Peers, Admission Peers and Member peers which have been labeled as CP, AP and MP respectively. Some control peers periodically monitor the Group Performance Index (GPI) which could be measured by factors like the current trust value of peers in the group, percentage of member peers and control peers in comparison with total group size and permissible joins as low-level peers within a time frame. If GPI falls below a certain threshold, a control peer measuring group behavior could trigger an internal event like a request for policy change which would be done in consensus with other control peers at that level and group would now go to state \( S_j \) with policy \( P_j \). Group events E1...En would continue to occur in this new state with variations decided by the new policy. An example of the need for change in state for F/OSS application would be, when with the current join policy, number of good peers with successful join in last 3 hours is less than 5 say. Thus state changes are triggered internally due to the effect of external events on the group.

Currently we define two levels of policies viz: Global Policies and Domain Specific Policies.

Global policy

This is the policy initiated by the creator of the group. It includes rules for admission, updating and leave as well as rules for deciding the no of control peers needed to collaborate to frame new rules, or change existing rules. Some sample rules could be:

- A peer can join as a member peer at level 1 of the group if he has received votes greater than 50% and if his current rating is greater than some threshold \( x \).
- To update level a peer must have an accumulated trust value \( > x \) and should be an authenticated member at the lower level.
- A new level could be introduced if at least \( y \) no. of peers are present in the current highest level and all 100% of them collaborate.
• For a new policy to be framed at least 75% of existing peers with trust value > some threshold x must collaborate. This ensures that malicious peers cannot frame new policies.

Modeling Rules for Join: An example of rules for peer with unique identity to join as a member peer in a group at level1 of a groups is 
\[ \text{verify}(\text{Npeer}, \text{Request}, \text{Rl}, \text{Level}, \text{Rate}, \text{Vote}) : - \text{assert}(\text{belongs}(\text{Npeer}, \text{Level})), \text{join}(\text{Request}), \text{member}(\text{Rl}), \text{Level}==1, \text{Vote} \geq 40. \]

Modeling Rules for Level Updating: When a member peer requests for level updating, the policy verifies the current level of the peer and checks his trust level.
\[ \text{verify}(\text{Npeer}, \text{Request}, \text{Rl}, \text{Level}, \text{Rate}, \text{Vote}) : \text{Level}1 \text{ is } \text{Level} - 1, \text{belongs}(\text{Npeer}, \text{Level}1), \text{update}(\text{Request}), \text{member}(\text{Rl}), \text{Level}==2, \text{Rate} \geq 4, \text{retract}(\text{belongs}(\text{Npeer}, \text{Level}1)), \text{assert}(\text{belongs}(\text{Npeer}, \text{Level})). \]

Domain specific meta policy

This set of policies inherits the properties of the global policy and contains application specific policies.

Modeling Meta Policies: We are able to express dynamic rules at runtime, as well as change existing policies depending on the status of the group at runtime by exploiting the assert and retract clauses of Prolog. A rule is asserted in prolog when all the clauses on the L.H.S of assert are true. Thus if a peer requests to join the group as a member peer when the number of member peers in the group is greater than 50% of current group size then the previous join rule is deleted from the database and a new join rule is asserted, thereby postponing low-level requests for join.
\[ \text{update_engine}(\text{Levels}, \text{MP}, \text{AP}, \text{CP}) : - \text{MP} > 50, \text{retract}((\text{verify}(\text{Npeer}, \text{Request}, \text{Rl}, \text{Level}, \text{Rate}, \text{Vote}) : - \text{join}(\text{Request}), \text{member}(\text{Rl}), \text{Level}==1, \text{Vote} \geq 40)), \text{retract}((\text{verify}(\text{Npeer}, \text{Request}, \text{Rl}, \text{Level}, \text{Rate}, \text{Vote}) : - \text{joint}(\text{Request}), \text{admission}(\text{Rl}), \text{Level}==1, \text{Vote} \geq 60)), \text{assert}((\text{verify}(\text{Npeer}, \text{Request}, \text{Rl}, \text{Level}, \text{Rate}, \text{Vote}) : - \text{joint}(\text{Request}), \text{member}(\text{Rl}), \text{Level}==1, \text{Vote} \geq 60)), \text{assert}((\text{verify}(\text{Npeer}, \text{Request}, \text{Rl}, \text{Level}, \text{Rate}, \text{Vote}) : - \text{joint}(\text{Request}), \text{admission}(\text{Rl}), \text{Level}==1, \text{Vote} \geq 80)). \]

IV. SIMULATION AND EXPERIMENTS

We chose the F/OSS application to demonstrate the behavior of P2P groups with dynamic policy changes and did some preliminary and simplistic experiments. A typical F/OSS application has core developers, reviewers, documenters, moderators. For simplicity we mapped core developers of code to Level 1, reviewers to Level 2 and Moderators to Level 3. A peer in Developer Level for e.g. would be updated to reviewer based on his performance and reputation earned in the group. At each level, Peers assume roles of Member Peers, Admission Peers and Control Peers based on the access policy. At level 1 member peers develop code, while control peers do additional administrative tasks.

We modeled the P2P groups using a java based overlay simulator namely PeerSim. The global and domain specific policies were implemented using SWI-Prolog which we integrated with Peersim at runtime wherein java and prolog talk to each other by means of sockets. We did runs with varying values of initial number of nodes and took sample runs for events where peers trigger events such as join, leave and update. The events have been triggered in the event mode of peersim.

A. Peer Behavior for Simulation

A peer declares an initial self proclaimed rating SPR along with his join request. The access policy allocates appropriate role and task to a peer based on his credentials, join-request and SPR. The peer is rated periodically by other peers based on his performance in the group, and his rating could reach the maximum intrinsic potential. We tested the framework by applying different job allocation policies. Graph in Figure 2 shows that the job success rate improves in a group which has a more stringent role and job allocation policy based on initial SPR of peer. Policy 1 allocates a complex job to a peer with lower initial SPR as compared to Policy 3. The framework permits the group to switch between the policies based on current group composition.

Graph in Figure 3 is a plot of the average current rating of a peer. Q indicates the current rating as computed by the group and SPR is the self proclaimed rating declared by the peer at the time of join. It is observed that for peers who over estimate their initial rating, the group eventually decreases their average trust value. Under estimation of initial trust value gives better performance than over estimation.

![Fig. 2. Job success rate versus time taken](image-url)
able to chose the right set of policies which permits the group to evolve to give its best output. Simulations show that as the group evolves with random runs of peersim events, the group switches between policies with changing group composition and trust levels. Thus when number of member developer peers exceed 50% of group size the group join policy tightens and a new join policy is selected. Similarly, when no. of control peers exceed a certain threshold as compared to member peers then a new join policy for control peers is selected. Similar changes occur for updating of levels.

V. RELATED WORK

Cassandra [12] is an authorization language that defines the actions of activating a role and deactivating a role. Users can thus write state-dependent and implicitly state manipulating policies, but this rather ad-hoc approach is inflexible. Some languages such as Ponder [13] support obligation policies. An obligation is a task to be executed after evaluating and enforcing an access request. However it does not provide a precise semantics for the state changes. Some work has been done on analyzing security properties in dynamic role-based systems, in the context of the role-based authorization language RT [14] and Administrative RBAC (ARBAC) [15], where members of administrative roles can modify the role membership and privilege assignments. In [11], policies written in Datalog can refer to facts in the authorization state, as in our model. Hezberg et al. propose in [9], a prolog-based trust management language, but do not focus on dynamically changing policies with the state of the environment.

VI. CONCLUSION AND FUTURE WORK

We have designed a framework for modeling dynamic policy driven trust based access control model in P2P applications. Peers are permitted to collaboratively frame and modify policies at various levels. They can also update their access rights within a group. Join priorities can also be decided by the group policy and depend upon current group composition. Changes to the policy itself can obviously affect the set of actions that are permitted or denied. Future work would include measuring the behavior of the group, and group evolution with varying group compositions and policies. Adaptive trust evolution in presence of strategically altering behavior of malicious peers is another interesting direction of work.

ACKNOWLEDGMENT

We would like to thank Ashish Arya for his contributions towards the simulations and experiments.

REFERENCES